T.C.
MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN
PURE AND APPLIED SCIENCES

A METHODOLOGY FOR CONSTRUCTIVE DEVELOPMENT OF A SYNTACTIC PATTERN RECOGNITION MACHINE (SPRM) USING REINFORCEMENT LEARNING PRINCIPLES

Fuat GELERİ
(Computer Engineering, MSc)

THESIS
FOR THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING PROGRAMME

SUPERVISOR
Assoc. Prof. M. Borahan TÜMER

İSTANBUL
2008

T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

A METHODOLOGY FOR CONSTRUCTIVE DEVELOPMENT OF A SYNTACTIC PATTERN RECOGNITION MACHINE (SPRM) USING REINFORCEMENT LEARNING PRINCIPLES

Fuat GELERİ, MSc

(141100320040258)

THESIS

FOR THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING PROGRAMME

SUPERVISOR

Assoc. Prof. M. Borahan TÜMER

İSTANBUL

2008

T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

ACCEPTANCE AND APPROVAL DOCUMENT

A METHODOLOGY FOR CONSTRUCTIVE DEVELOPMENT OF A SYNTACTIC PATTERN RECOGNITION MACHINE (SPRM) USING REINFORCEMENT LEARNING PRINCIPLES

Established committee listed below, on .... and .... by the INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES' Executive Committee, have accepted Mr. Fuat GELERİ 's Master of Science, titled as "A Methodology for Constructive Development of A Syntactic Pattern Recognition Machine (SPRM) Using Reinforcement Learning Principles" in Computer Enginering

COMMITTEE

Assoc. Prof. M. Borahan TÜMER (Thesis Supervisor)	. . . .
Prof. M. Akif EYLER	. . . .
Prof. Fuat İNCE	. . . .

Date of thesis' defense before the committee: . . . .

APPROVAL

Mr. Fuat GELERİ has satisfactorily completed the requirements for the degree of Master of Science in Computer Engineering at Marmara University. The Executive Commitee approves that he be granted the degree of Master of Science on ....
(Resolution no: ....)

DIRECTOR OF THE INSTITUTE
Prof. Dr. ....

ACKNOWLEDGEMENTS

First, I want to thank my advisor Assoc. Prof. M. Borahan TÜMER for his guidance and support in development of thesis application and in the completion of this thesis.

Secondly, I thank all my friends for their understanding. Special thanks go to my friends Çetin MERİÇLİ, Gülefşan BOZKURT, Murat BALABAN and Ülker ÖZGEN for their great support and understanding.

Finally, it is my pleasure to thank my family, Ayşe GELERİ and Aydın GELERİ for their endless support and belief in me and my work.

TABLE OF CONTENTS

ÖZET

PEKİŞTİRMELİ ÖĞRENME KURALLARINI KULLANARAK SÖZDİZİMSEL ÖRÜNTÜ TANIMA MAKİNESİ (SPRM) TASARIMI

Sözdizimsel örüntü tanıma yapay zekanın günümüzdeki en önemli parçalarından biridir. Sonlu özdevinirin bulunması örüntü tanıma alanındaki ilerlemelere hız kazandırmıştır. Örüntü tanıma sorunlarının çoğu için özdevinir elle kurulmuş ve ayarlanmıştır. Özdevinirin otomatik elde edilmesi, gürültülü ve eksik veriler ile başa çıkmayı sağlarken, işlemi, elle kuruluşu gibi, kullanışsız ve zaman alıcı bir iş olmaktan çıkarır.

Bu çalışma gürültülü ve eksik veriler içeren sinyal serilerinin incelenmesi için, sağlam ve uyarlanabilir bir araç tasarlamayı hedeflemektedir. Çalışma, alfabenin neden olabileceği her diziyi tanıyabilecek bir özdevinir ile başlar, ve gittikçe, daha özel serileri kabul edecek özdeviniri bulmayı hedefleyen bir arama algoritması gibi devam eder. Öğrenme devam ettikçe, özdevinir, sunulan veri dizisindeki gizli örünüye daha da çok benzeyerek, arama algoritmasının bir genel minimuma doğru yaklaşmasına neden olmaktadır.

Yöntem, değişen markov süreçlerin öğrenilmesini ve yakın periyotların ayrılmasını başarıyla sağlamıştır ve ortalama gürültü seviyesi için yüksek başarımlar göstermiştir. Gürültü seviyesinin artması, kullanılan alfabenin az elemanlı olması, ve periyot boyunun uzaması başarı oranının düşmesine neden olmaktadır.

Aralık, 2007 **Fuat GELERİ**

ABSTRACT

A METHODOLOGY FOR CONSTRUCTIVE DEVELOPMENT OF A SYNTACTIC PATTERN RECOGNITION MACHINE (SPRM) USING REINFORCEMENT LEARNING PRINCIPLES

One of the today's most important titles in artificial intelligence is syntactic pattern recognition. Development of finite automata accelerated the progress in this field. Automatical construction of the automaton solves the problems requiring dealing with a diversity of noisy and imperfect structures of data, which are cumbersome and time consuming when performed manually.

This work tries to provide tools that are robust and adaptive in analysis of noisy and imperfect signal sequences. Learning starts with an automaton that accepts every sequence that the used alphabet can lead, and continues like a search algorithm to find more and more specific automatons. As learning continues the automaton resembles the specific pattern that is hidden in the presented data sequence.

The method learns multiple-degree markovian processes and classifies very similar periods successfully, and shows high success rates for moderate noise levels. The success rate of the algorithm decreases as the noise rate increases; also the used alphabet size decreases and the length of the period increases.

December, 2007 **Fuat GELERİ**

CLAIM FOR ORIGINALITY

A METHODOLOGY FOR CONSTRUCTIVE DEVELOPMENT OF A SYNTACTIC PATTERN RECOGNITION MACHINE (SPRM) USING REINFORCEMENT LEARNING PRINCIPLES

In syntactic pattern recognition, the primary aim is construction of an automaton that is able to deal with imperfect and noisy information. In many works this construction is performed manually, and generally the automatical learning algorithms stay in learning the first degree markovian processes. The novelty of the approach is that the learning method that creates the automaton is capable of recognizing multiple-degree markovian processes with a considerably high accuracy for reasonable levels of noise. The learning algorithm uses principles of reinforcement learning and neural networks and models the syntactic pattern recognition process as a search problem.

December, 2007 Assoc. Prof. Borahan TÜMER Fuat GELERİ

LIST OF SYMBOLS

$\|C\|$	cycle length
$\|\boldsymbol{\Sigma}\|$	alphabet size
1-D	one dimensional
2-D	two dimensional
3-D	three dimensional
$\mathbf{c}$	a matrix of response probabilities
$\mathbf{C}$	cycle
$\mathbf{D}_{ij}$	any token sequence
$\mathbf{F}(.,.)$	a stochastic transition function
$\mathbf{H}(.,.)$	a stochastic output function
$\mathbf{L}$	learning algorithm
$\mathbf{L}_{R-I}$	learning algorithm based on linear reward-inaction scheme
$\mathbf{O}(.)$	average runtime function
$\mathbf{Pr}(.)$	probability of state or transition
$\mathbf{Q}$	periodic sequence
$\mathbf{S}$	number of states
$\boldsymbol{\alpha}$	a set of actions
$\boldsymbol{\beta}$	a set of environment responses
$\boldsymbol{\delta}_{\varphi_i\varphi_j}^{\tau_k}$	transition (from $\boldsymbol{\varphi_i}$ to the $\boldsymbol{\varphi_j}$ with $\boldsymbol{\tau_k}$)
$\boldsymbol{\delta}^{+}$	selected transition
$\boldsymbol{\Delta}$	set of transitions
$\boldsymbol{\Delta}^{+}$	added set of improbable transitions
$\boldsymbol{\Delta}^{-}$	removed set of imrobable transitions
$\boldsymbol{\varphi}$	state
$\boldsymbol{\varphi}^{+}$	added improbable state
$\boldsymbol{\varphi}^{-}$	removed improbable state
$\boldsymbol{\Phi}$	set of states

$\boldsymbol{\lambda}$	constant learning parameter
$\boldsymbol{\vartheta}_{\boldsymbol{\varphi}_i}$	state frequency
$\boldsymbol{\vartheta}^{\boldsymbol{\tau}_k}_{\boldsymbol{\varphi}_i\boldsymbol{\varphi}_j}$	transition frequency
$\boldsymbol{\Sigma}$	token alphabet
$\boldsymbol{\tau}$	token

LIST OF ABBREVIATIONS

ASPRM	Adaptive Syntactic Pattern Recognition Machine
CAS	currently active state
CNC	cycle neighborhood coefficient
ECG	Electrocardiogram
EEG	Electroencephalogram
FFT	Fast Fourier Transform
FSLA	Fixed Structure Learning Automaton
GA	Genetic Algorithms
LA	Learning Automata
MCL	maximum cycle length
NAS	next active state
NN	Neural Networks
OCR	Optical Character Recognition
RBF	Radial Basis Function
SNC	state neighborhood coefficient
SR	Speech Recognition
TNC	transition neighborhood coefficient
VSLA	Variable Structure Learning Automaton
XML	Extensible Markup Language

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND OBJECTIVES

1.1. INTRODUCTION

Today we see applications of artificial intelligence in many areas of our life. Increasing recognition capability leads to more powerful algorithms. In artificial intelligence pattern recognition seems one of the most important titles. As the recognition capability increases, decision-making algorithms make decisions with more dependable knowledge. Knowledge is extracted from the data collected by sensor devices from the environment. Pattern recognition algorithms are utilized in extraction of knowledge, like an object in a picture, a structure in a sequence, etc. With the invention of automaton, design of a recognition system, which was requiring involvement of high quality personel, became a daily task [1].

Current advances in the technology provide engineers the power of performing more difficult tasks. This capability leads to big improvements in the recognition of human face, handwritten text, speech, or fingerprint problems. In most of the proposed solutions, pattern recognition techniques are used. Syntactic approaches require detection of patterns from a given relevant data [2, 3, 4]. According to the target problem the structure of the raw data shows variances. In some problems we use $1-D$ sequence, for example medical recordings received from various parts of the human body (ECG, EEG recordings). Whereas, in recognition of human face or fingerprint the raw data are the $2-D$ images. In such cases we use graph or tree like structures.

1.2. OBJECTIVES

It is shown that, using the principles of reinforcement learning, VSLAs can be devised with a constructive approach to detect cycles or repetitive patterns in noisy periodic or semi-periodic sequences. [5]

Previous work in this topic [5] requires some constants like Maximum Cycle Length (MCL), Cycle Neighborhood Coefficient (CNC), State Neighborhood Coefficient (SNC) and Transition Neighborhood Coefficient (TNC) to be determined manually to construct the automaton by detecting and eliminating the noise. In this algorithm the best possible values for these constants are found by “brute force” like approaches.

Furthermore, to find the correct sequence the Cycle Detector requires presence of clean cycles in the noisy sequence. The algorithm fails in some special cases that makes it impossible for the Noise Remover to distinguish between two different cycles.

In this work we will try to decrease the count of used constants, find solutions for the cases, which the previous work fails to solve, and improve the algorithm to work for signals those have structures from multiple-markov degrees.

2. GENERAL BACKGROUND

2.1. LEARNING AUTOMATA

Systems generate huge amount of data. These data contain some valuable information, some noise, and some unimportant sections. Generally the generated raw data is not totally random. Systems operate on a logic and create these data. Sometimes identifying the complete logic of a system may not be possible, still certain patterns or regularities with some accuracy can be detected [6].

In many fields we see application of machine learning principles. Banks analyze their past data to use in credit applications, fraud detection and the stock market. Learning principles are used for optimization, control and troubleshooting in manufacturing. Learning programs are used for medical diagnosis. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service. Large amounts of data in physics, astronomy and biology can only be analyzed fast enough by computers. Searching for relevant information cannot be done manually in the internet [6].

All the systems show some changes as the time elapses. Machine learning algorithms should have the ability to learn and adapt to the changes. With such an algorithm the designer does not have to worry about each possible case that environment can produce.

Combined work of psychologists (modeling observed behavior of being known to be intelligent like humans and animals), statisticians (modeling the choice of possible actions based on past observations and tryouts), operations researchers (implementing optimal strategies) and system theorists (finding rational decisions in random environments) constitute the concept of Learning Automata (LA) [2].

Initially Tsetlin introduced the concrete, analytical concept. Varshavskii and Vorontsova continued the study of learning behaviors and abilities of automata. Studies in this topic have been done extensively by many researchers [7].

Giving some definitions about LA operating in an unknown environment will make our understanding more clear. All external conditions affecting the automaton are the environment. Environment can be defined by a triple α, c, β. α represents a set of inputs, c a set of penalty probabilities and β a set of outputs (2.1).

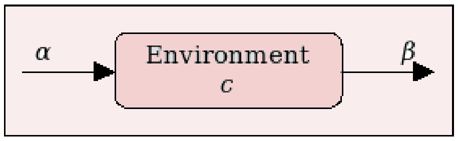


Figure 2.1. Environment in LA.

Application of input $\alpha(n) = \alpha_j$ to the environment is done at discrete time steps n (n $= 0, 1, 2, \text{inf}$). Environment generates a response $\beta(n) = \beta_i$ for each action by utilizing c as $c(\alpha, \beta) = c_{ij}$.

The concept of an automaton used in automata theory is a very general one applicable to a variety of abstract systems. Automaton is defined by the quintuple:

$<\Phi,\ \alpha,\ \beta,\ F(.,.),\ H(.,.)>$ where

* $\Phi = \varphi_1, \varphi_2, \ldots, \varphi_S$ is a set of states of size S,
* $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_r$ is a set of output actions of size r,
* $\beta = \ \beta_1, \beta_2, \ldots, \beta_q$ is a set of inputs of size q,
* $F(.,.) : \Phi x\beta \rightarrow \Phi$ a transition function that maps current state and the current input to the next state,
* $H(.,.) : \Phi x\beta \rightarrow \alpha$ an output function that maps current state and the current input to the current output.

In this automaton the input and current state determine next state and output action. Such an automaton is called finite if sets Φ, α and β are finite. Parameters of both functions $F(.,.), H(.,.)$ are state and automaton input, and the outputs state and action respectively. Those functions determine their outputs using $3 - \textit{dimensional}$ matrixes where first two dimensions are states and inputs, and last dimensions are states and output actions respectively. All entries of those matrixes are values in $[0,1]$ and stand for probabilities. For example, to calculate an output function $F(.,.)$ selects an action that corresponds to the maximum value in the array $F[\varphi_i, \beta_j]$ in the matrix $F[\alpha, \beta, \Phi]$ where i is the current state and β_j is the current input. This is same with function $H(.,.)$.

If the entries of transition and/or output function matrixes change with time to improve performance, this automaton called learning automaton because the result of mentioned functions may not always be the same.

The connection of a learning automaton to the environment is shown in Fig.2.2 The output of an automaton at the time n is an action $\alpha(n)$ which at the same time is an input to the environment. In its turn, the response of an environment $\beta(n)$ is an input to an automaton.

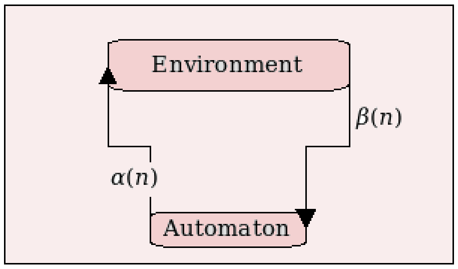


Figure 2.2. Learning automaton and the environment.

2.2. REINFORCEMENT LEARNING

We have various machine learning types, the most common types of learning is supervised, unsupervised and reinforcement learning. According to the input at our hand we use one of the methods in these three categories.

If the data contain both the input values and the corresponding output values for each than the type of learning function is supervised learning [6]. The aim of learning is finding F(.,.) and H(.,.) function matrices from the given input output mappings.

In unsupervised learning we do not have any information about the corresponding output of an input. Instead of trying to find a mapping function from the inputs to the outputs, we search the similarities between the inputs. So we generate clusters from the inputs. The best example of unsupervised learning is lossy compression algorithms [8]. Patterns occuring more frequently than others will be the main properties of the clusters. In statistics this is called frequency estimation [6].

In reinforcement learning there is two main actor, an agent, which is the intelligent algorithm, the body, making the calculations, and artifical modellings, and giving decisions, and the environment, which has rules, and according to these rules, and according to the behavior of the agent gives some reward or punishment to the agent. Reinforcement learning is learning what to do to maximize the taken total reward amount. The agent takes some actions, and the environment provides some reward or penalty values for each action. Environment may be deterministic or non-deterministic. In deterministic case environment can be modelled as having a number of states, and various actions available at each state. The goal of the agent is to correctly model the environment so as to know the actions to take in each state that will maximize the taken total reward amount. Because the agent tries to maximize the total reward amount, the agent should try to reach “high-valued” states that provide relatively high rewards. So the states on the path to “high-valued” states should also receive higher frequency of selection. To achieve these various reinforcement learning algorithms are modelled [3]. Eligibility traces, Sarsa algorithm, Q-Learning and POMDPs are some of the reinforcement algorithms [3, 9].

The agent should take the actions that provide the best reward values. However, the agent should also explore for new states and actions following the possibility that unexplored state and action may provide a much higher reward. So, the agent should exploit already known actions and it should also explore in order to find better actions in the future.

In reinforcement learning we have some terms like: an agent, an environment, a policy, a reward function, a value function and a model [10]. We have discussed agent, which is an automaton, and environment before. A policy is an agent's set of rules by which the agent behaves. The agent uses the policy to move to its next state and take its next action, given the current state and the response of an environment. In other words, a policy is F(.,.) and H(.,.) functions together. It is usually a set of matrices of stochastic values, and sometimes it may require extensive search and computation to achieve this matrices [6, 10].

An agent's objective is to maximize the received reward in the long run. Agent models a reward function that defines what states, transitions and actions are good and bad for the agent according to the response received from the environment.

A reward function shows the currently favorable action and state, however the favorable action and state in the long run are shown by the value function. A value function calculates a total amount of reward that can be expected starting from given state. A state might always receive low immediate reward, but still have high value because it is followed by states receiving high rewards. According to the sequences of agent's observations the value function is calculated, and it shows the quality of the policy.

Model in the reinforcement learning represents the behavior of the environment. A model can provide the prediction about the next state, reward and action given current state and the response value. Reinforcement learning algorithms uses the model to select the next action and it considers number of possible situations before they actually experienced.

2.3. SYNTACTIC PATTERN RECOGNITION

For longer than three decades syntactic pattern recognition has been popular and widely applied to many life recognition problems. It continues to be used widely in many areas. Some application areas are optical character recognition [11], fingerprint recognition [12], speech recognition [13], remote sensing data analysis, biomedical data analysis [14, 15, 16], scene analysis [17], texture analysis [18], three-dimensional object recognition, two-dimensional mathematical symbols [11], and geophysical seismic signal analysis [19, 20].

In pattern recognition problems we use some structural information, lists, graphs, trees, matrices, etc to describe the pattern. A pattern can be decomposed into more and more simple subpatterns. The simplest subpatterns are called primitives or terminals. Via a parser, these primitives or terminals are parsed to be assigned to the correct class [4].

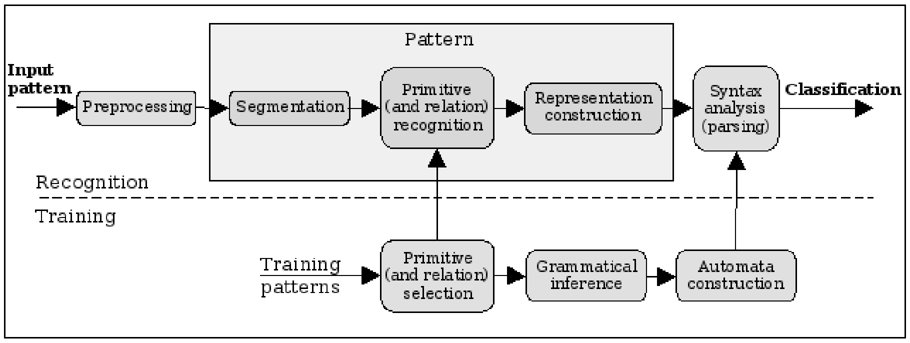


Figure 2.3. The flow of syntactic pattern recognition process.

We call the process of parsing the primitives to generate the patterns as pre-processing. In this part the raw input is filtered, grouped, classified and compared with common patterns, so each primitive or primitive group is assigned to classes with certain accuracies.

3. ADAPTIVE SYNTACTIC PATTERN RECOGNITION MACHINE

In real life there are many phenomena which are periodic. One of them is the heart beat. This periodic data flow carries some vital information about the health of the heart. Healty heart's ECG output differs from the ECG output of unhealty heart. So, if we can model this periodic behavior, we may achieve a model for healty hearts, and this model can be used in classification of illnesses on the heart.

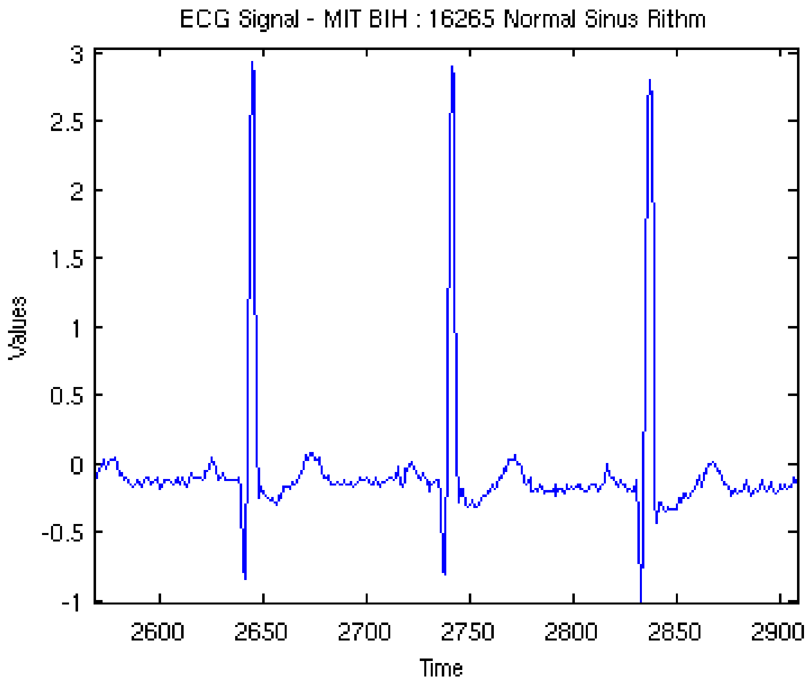


Figure 3.1. Example ECG signal, normal sinus rithm

3.1. PRE-PROCESSING

First of all lets investigate the periodic behavior of a heart signal. Figure 3.1 shows an example ECG signal. When we take FFT (Fast Fourier Transform) of a normal sinus rithm, we achieve such a figure in the frequency domain, shown in Figure

3.2.

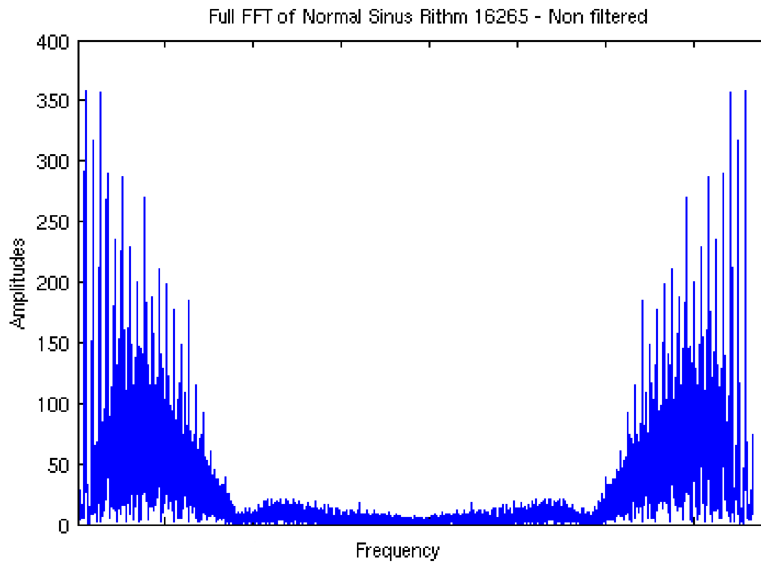


Figure 3.2. Fourier Transform of a sormal sinus rithm

As seen in Figure 3.1 and Figure 3.2, the original signal carries high noise. The FFT graph shows peaks in the high frequency domain. Therefore we have to apply some filters on the signal. For this purpose we applied unweighted sliding-average smoothing method 3.1,

$$S_j = \frac{Y_{j-1} + Y_j + Y_{j+1}}{3} \tag{3.1}$$

and also triangular method 3.2, as seen in Figure 3.3.

$$S_j = \frac{Y_{j-2} + 2Y_{j-1} + 3Y_j + 2Y_{j+1} + Y_{j+2}}{9} \tag{3.2}$$

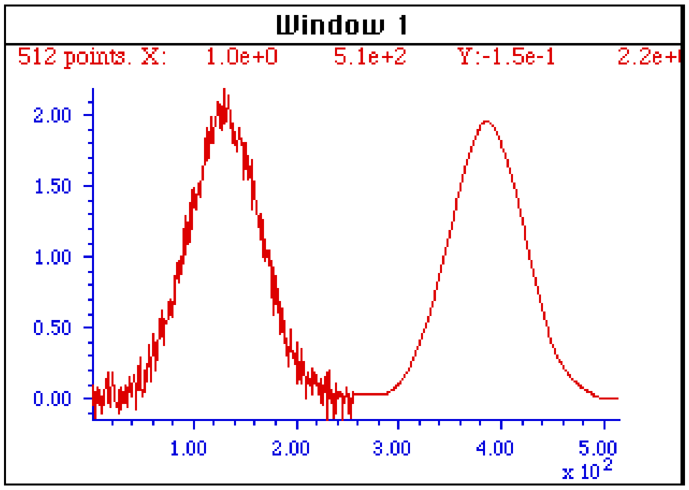


Figure 3.3. Result of unweighted sliding-average smoothing and triangular smoothing methods. Picture is taken from [21]

Moreover, we tried to fit lines to the ECG signal. An example line fitting is shown in Figure 3.4.

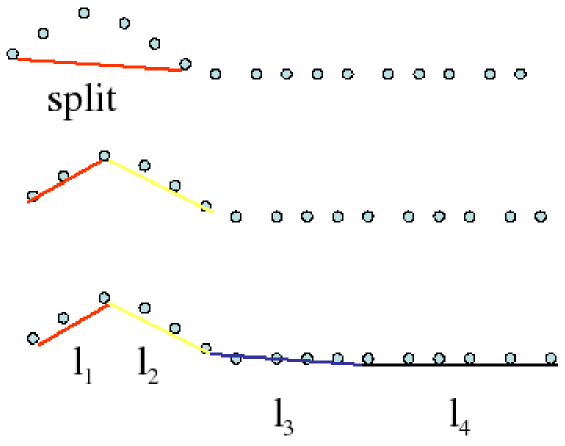


Figure 3.4. Line fitting to a signal

We also used some advanced filters like Wiener filter [22]. As seen in Figure 3.5, the result is quite satisfactory. It nearly eliminated all noises, and an optimal ECG signal is achieved from a normal sinus rithm. The data is from MIT BIH Normal Sinus Rithm database, number 16265.

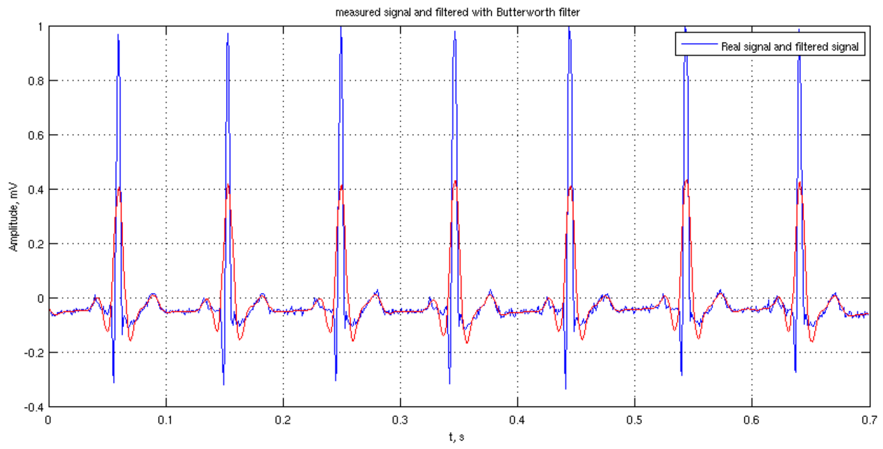


Figure 3.5. ECG signal filtered with Wiener filter

The filter removed the high frequency components, as seen in Figure 3.6, so it eliminates the noises.

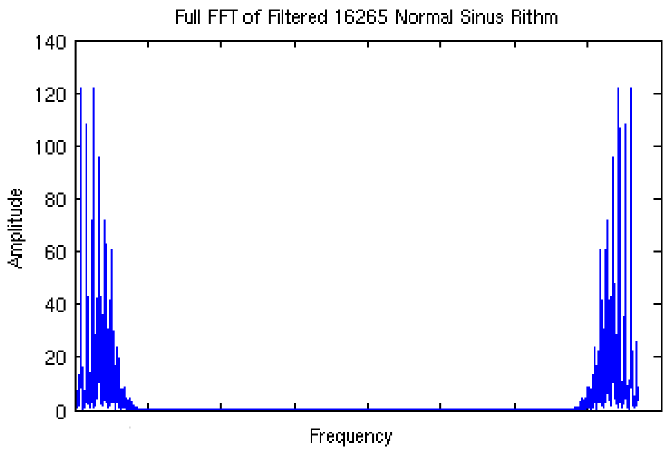


Figure 3.6. Frequency analysis of filtered ECG signal

When we apply inverse FFT to the FFT of filtered original signal after removing the high frequency components, the result is a clear ECG signal, shown at Figure 3.7.

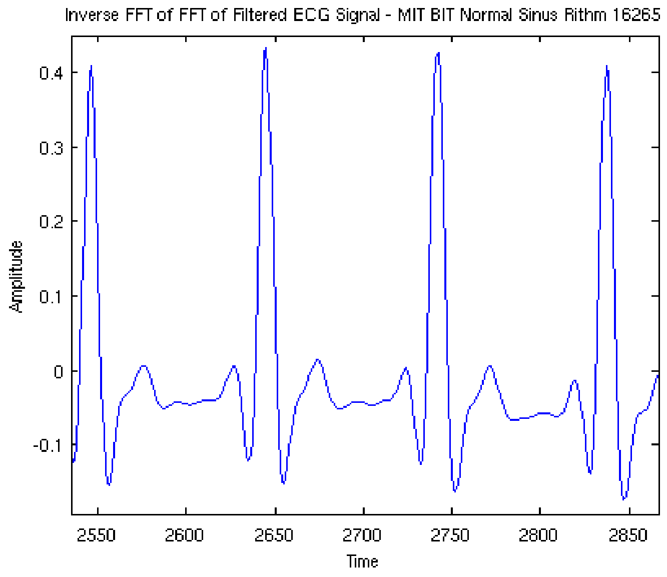


Figure 3.7. Inverse FFT of FFT of filtered ECG signal

As seen the better the filter, the better the output is. When a perfect filter is applied, just like the Wiener Filter, the output will be near to the synthetically produced signal, shown at Figure 3.8. So in our tests this signal is used.

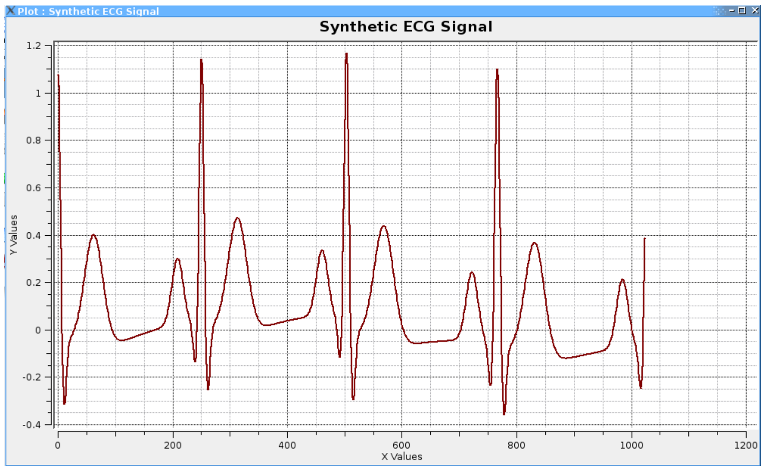


Figure 3.8. Synthetically produces signal

3.1.1. K-Means on the Signal

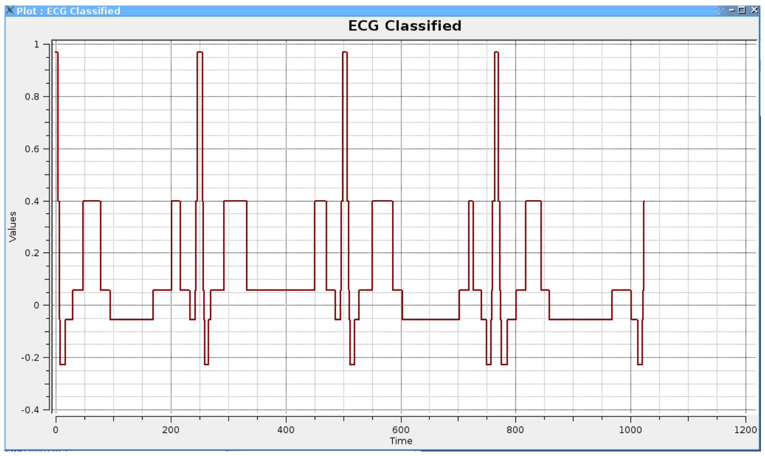


Figure 3.9. Classified filtered ECG signal

We applied K-Means on this signal. As seen in Figure 3.9 the P and T sequences are very near to each other. So the output of K-Means bring much more complexities. Moreover, we cannot further simplify, and reduce mean count to escape from this mixing.

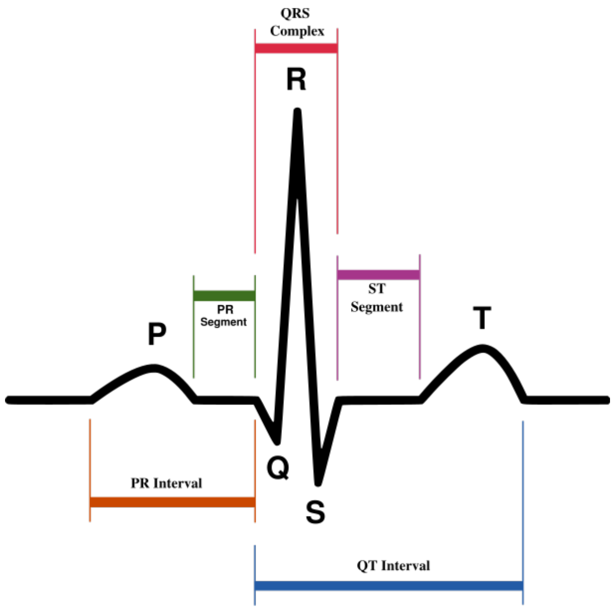


Figure 3.10. Model of heart signals. Picture is taken from Wikipedia [23]

In fact we may simplify the system further by combining these two parts P and T, so this may solve the problem. However as seen in Figure 3.10, the components of ECG signal will be lost. From the Figure 3.10, we understand that our filter worked very well.

The result of applying K-Means on the filtered ECG with six as the **k** value results to -0.0607, -0.131, 0.0023, 0.1841, 0.3621 and -0.0355. The values -0.0607, 0.0023, and -0.0355 are very near to each other. So we combined them to result 0.0 as the mean. So the resultant mean values are 0.0, -0.131, 0.1841, and 0.3621. If we look at the figure 3.10 again we assumed P and T sequences as the same, also Q and R sequence is also assumed the same.

As seen the resultant sequence is like I, P, I, Q, R, Q, I, T, where I is the mean, valued 0.0. This decreases the classification capability of the resultant automaton. So a better approach should be developed to generate the feature space to be used in the learning. Which may be syntatically representing the P, Q, R, S, T waves also.

So, modelling each primary event should be the main preprocessing step. However, we could not find chance to further investigate this step. As a future work we will work on this.

3.2. THE PROBLEM STATEMENT

When we look at the structure of a period, we see repeating patterns, and a sequence. If we name each significant part, as seen in Figure 3.10, the sequence is like P, Q, R, S, T. In fact the literature does not count the stable part of the signal, which is nearly at the 0.0, which we name **I**. So the sequence is in fact like I, P, I, Q, R, S, I, T. A quite simple sequence for human to catch. However, it important to classify the segments of the signal as P, Q, R, T, I, with some low or moderate error rates.

We further investigated the syntactic possibilities a period can carry, and tried to design a general algorithm, that will cope with various problems, not only the problems in the ECG signals.

One of the problems is continous learning until finding a single sequence. Namely the algorithm should not only learn the probabilities of the transitions, as in the Hidden Markov Models, but also learn the model. Yet another problem is the "OR" problem, two periods comming after each other in a random order. The algorithm should also be capable of dealing with multiple degree markov processes to learn various length of segments in a period.

In this purpose, instead of working with signals we used alphabet elements. Each alphabet element represents a different group in the original input. Let me clarify this. If we look at a handwritten text, we may see many "a" characters. They may be totally different from each other, but they are all "a". So with some error rate, instead of using different mean values, we assume all of them to be in the group of **a**. As one of the mean values in this group is observed, group "a" is used instead.

So the groups are the alphabet of our learning process. Any combination of each alphabet element constitutes the full universe. The universe, which we can represent by the alphabet.

For example, if the alphabet consists of **a**, **b**, and **c** tokens then full universe can be shown as **(a*b*c*)***. This expression can accept any combination of a, b, and c. Our aim is finding a minimum from this full universe.

At this minima, if it is the global minima, namely the correct sequence, the error rate will be at the minimum. So we may think of learning as a search of a global minimum, or acceptable minima.

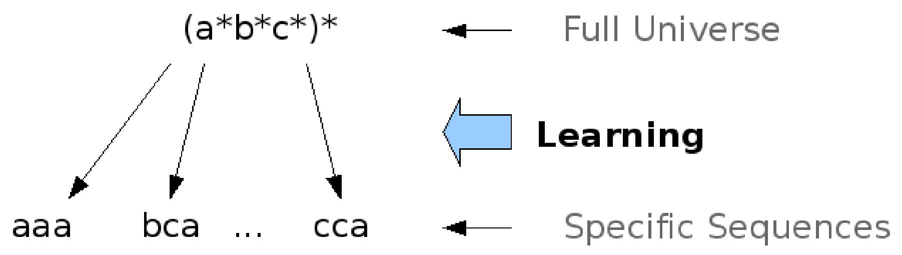


Figure 3.11. Learning is a search for the specific sequence from a full universe.

The full universe keeps all possible sequences that the alphabet can lead. So we model it as a fully connected graph of all alphabet tokens. Each token has also a self loop. Fig. 3.12 shows the full universe of the alphabet consist of three tokens, “a”, “b”, and “d”.

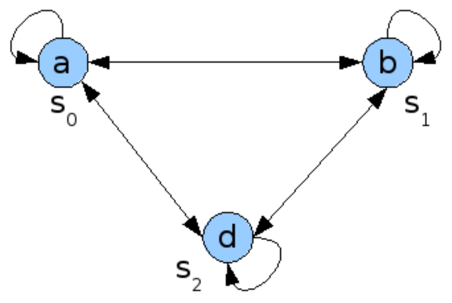


Figure 3.12. The full universe for the alphabet with three tokens.

Because the automaton represents the whole universe, value of sum of all entropy values on its transitions is quite high. The algorithm should lead some transitons and nodes to be removed, to be split, and to be grouped together.

The automaton is traversed with an input sequence. The traversal results some transitions and nodes to acquire high probabilities, whereas others have less. Therefore, at the start, we have an automaton which has the capability of classifying the input sequences with limited success rates. The initial automaton accepts any kind of sequence generated with the tokens of the same alphabet. So the classification is based on the difference of the probabilities on the transitions and the states, this is not enough though. However, what should be the way to improve the classification power of the automaton?

If we inspect Fig. 3.12 again, we see that the automaton has only one instance of each token. So the automaton keeps information about only with which probability one token will come after the other. Level of predicting the next tokens is limited to one, at the start. The aim of learning should be increasing this level and going toward finding a single sequence if possible.

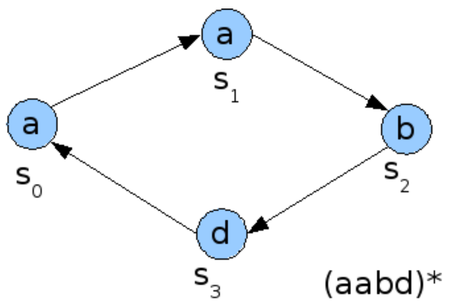


Figure 3.13. Absolute end of learning, a specific sequence is found.

A specific sequence will have no entropy, and no need to keep probability values at all. Each state will be visited once at each cycle. So if the algorithm leads to such a sequence this sequence should be the global minima of the learning.

A simple sequence $(aabd)$ can be represented by $(a*b*d*)*$, but $(a*b*d*)*$ can also represent many other various sequences. So we require an algorithm which will lead $(a*b*d*)*$ to $(aabd)*$, which is a kind of search algorithm.

Artificial intelligence methods, those searching global minima, e.g. gradient descent like algorithms, may get stuck into **local minima**. The same problem is valid for the search of the specific sequence. In such a case, a higher step is taken in the gradient descent algorithms to save from local minima. When this bigger step is taken, the next error value will be bigger, but this will lead the consequent error values to be smaller and smaller.

The same principles are applicable for the adaptive syntactic pattern recognition machine. The algorithm will remove some transitions and states to reach to a minima. Removal of the transitions with less than a threshold value, and the states with no incomming or outgoing transitions will decrease the entropy, and will lead a more specific automaton. So, when a transition or a state is removed, the automaton converges to a specific automaton. At a point, all the transitions will have probability values higher than the threshold value. However, the entropy of the automaton will not be equal to zero. We call this automaton, the local minima automaton. In some cases, automaton in the local minimum will be more robust to the noise. However, if a more specific automaton is wanted the learning should continue. In ASPRM, the algorithm saves from local minima by splitting a state. The state with the highest error, or the state leading to the highest entropy addition may be split.

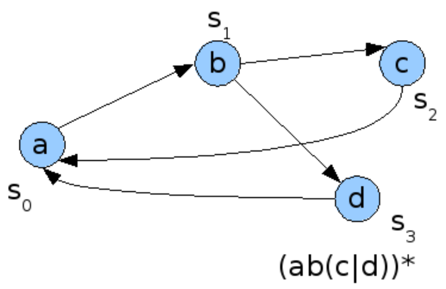


Figure 3.14. The automaton stuck into a local minima.

When a state is split, the entopy of the automaton is increased. However, this transformation leads the search to continue and it saves the automaton from the local minima. We may think the removal process as the exploitation and splitting as the exploration steps of reinforcement learning algorithm.

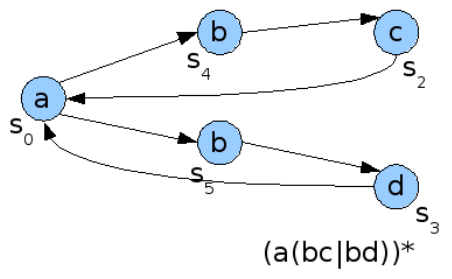


Figure 3.15. The state s_1, which is the source of highest entropy, is split, s_4 and s_5 is generated

Figures 3.15 and 3.16 shows two different automatons, but those two represent the same universes. Figure 3.15 has an automaton with five states, that is the same automaton with Fig. 3.14. The automaton is the saved version of the previous automaton which was stuck in local minima. However, the resultant automaton is also stuck in local minima. We split state s_1 and achieved another automaton. Figure 3.16 shows the resultant automaton which has six states and it is the saved version of the automaton in Fig. 3.15. Now learning can continue for this new automaton to achieve a more specific automaton.

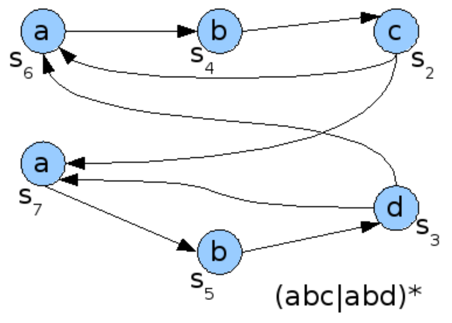


Figure 3.16. The state, s_0, which is the source of highest entropy, is split, s_6 and s_7 is generated

So the basic job of the algorithm is making search from a full universe toward a specific sequence, while escaping from local minima. In this purpose, we use reinforcement learning principles, some formulas from hidden markov models, and neural networks.

Furthermore, Genetic Algorithms is also used in generation of similar but different input sequences for test purposes.

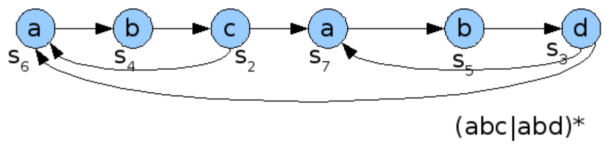


Figure 3.17. A better view of the same automaton.

As all of the learning steps indicate, learning the sequence is more important than finding the state and transition probabilities. The probabilities are only statistical information. The algorithm should be the way to use the statistical information intelligently to lead to the correct sequence.

Next, the difficulties, algorithm must cope with is introduced. One of them is the increasing affect of noise when the length of the sequence grows. The other difficulty is about periods having multiple degree markovness.

3.2.1. Adverse Effect of the Noise

The previous section described the steps of the learning. In each step, the algorithm uses statistical information to make decisions. Decision of whether to remove a transition, or a state, split a state, or group the states will depend on this statistical information. Therefore, the correctness of the statistical information is an inportant factor of success.

In signal processing, and in every area of pattern recognition, noise is inevitable. Learning algorithms should make the required generalization and robust to noise as much as possible.

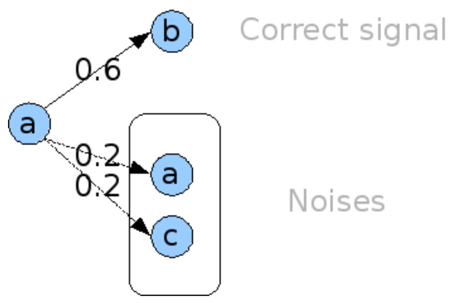


Figure 3.18. For a single token, the effect of the noise is not curicial.

Figure 3.18 shows the effect of noise for a single token. As seen, it is easy to cope with the noise on a single token, especially if the noise is distributed quite uniformly between other patterns. However, if the noise yields toward a single pattern, it will be more difficult to differentiate noisy and the real signal.

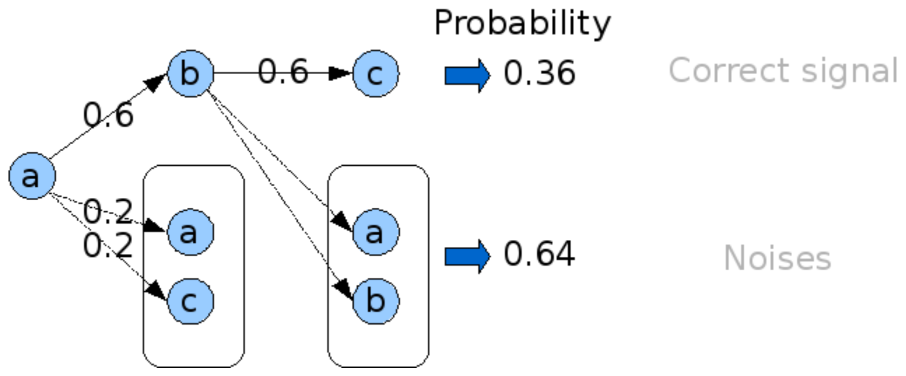


Figure 3.19. The effect of the noise increases exponentially for sequences.

In syntactic pattern recognition, the effect of noise in one pattern will grow exponentially while composing the syntactic pattern as shown in Fig. 3.19. So, if probability of getting noisy signal next is P_n, the probability of getting the correct signal in the next step will be $1-P_n$. If thought for a signal sequence of four tokens, the probability of having the correct sequence in the next four input pattern is equal to $(1-P_n)^4$.

The effect of noise in searching for a sequence is higher than expected. To make the algorithm, and the automaton more robust to noise some parts of the automaton should be kept ambiguous. It is difficult to have a single long sequence to get success, however to learn better, it is required to have longer single path sequences.

The noise ratio may vary from one signal set to another. According to the noise threshold the algorithm may see some transitions or states with low probability values as noisy or some noisy transitions as valid. So the decision of noise level should be provided by the user.

3.2.2. Multiple-Degree Markov Processes

In the previous work [24], it is possible to get the same VSLA's for two sequences with different cycles. In certain cases, if the number of used alphabet tokens are small, the repetition of the tokens in the cycle increases and the possiblity of getting the same VSLA's increases.

Let's inspect the problem as described in the previous work [5], and then describe our solution for the problem.

As a special case for recognition, the cycles $C_1 = (12131312)$ and $C_2 = (1213)$ are used. Noise-free (for simplicity) VSLA that is a result of VSLA Constructor is shown in Fig. 3.20. In the algorithm VSLA Construction process is based on $1st$-degree Markovian process where each token depends only on the previous token.

The algorithm keeps the automaton in a different structure than we do. We keep the tokens on the states, and transitions keep only the statistical information. However, in previous work, states keep only statistical information and the transitions keep the tokens. Here, the 'a', 'b', 'c' are the names of the states, 1, 2 and 3 are the tokens on the transitions.

So as seen in the Figure 3.20, to pass from state **a** to state **b** next input should be 1. As we traverse the automaton, starting at state **a** for input $1, 2, 1, 3$, the order of active states will be like **a, b, a, c, a**. However, when the input is $1, 2, 1, 3, 1, 3, 1, 2$, the active states will be seen in order, **a, b, a, c, a, c, a, b, a**.

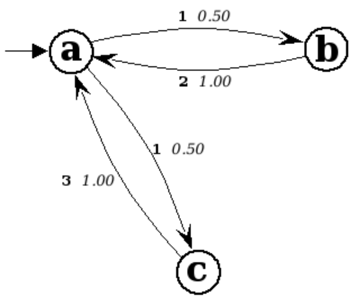


Figure 3.20. Structure of noise-free VSLA for cycles $C_1 = (12131312)$ and $C_2 = (1213)$.

As shown in the Fig. 3.20, the algorithm generated the same VSLA for two different cycles. For this problem they stated that "Although $1st$ degree Markovian process is enough to recognize almost all cycles, in these cases a $2nd$-degree Markovian process is required to distinguish between these two cycles. Increasing degree of Markovian process results in slight increase of recognition probabilities and exponential increase of runtime which is $O(|C|^d)$, where $|C|$ is the length of the cycle and **d** is the markov degree". It is proposed that, the usage of $2nd$-degree Markovian process is enough for the solution of this problem, as seen in Fig. 3.21 and Fig. 3.22.

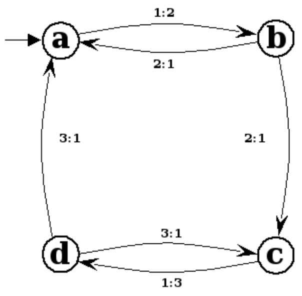


Figure 3.21. VSLA, constructed using $2nd$ degree Markovian process, for $C_1 = (12131312)$. Figure is taken from the thesis of Aleksei Ustimov.

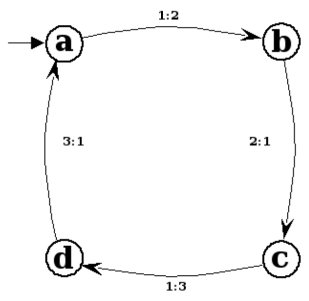


Figure 3.22. VSLA, constructed using $2nd$ degree Markovian process, for $C_2 = (1213)$. Figure is taken from the thesis of Aleksei Ustimov.

So the solution proposed by the previous work is making transitions to carry two token information. The information is both about the current token and the next token. For example "1 : 2" means, this transition will be taken with input "1" and after taking this transition the next transition will be taken by input "2". So if the next inpur is not "2" the transition will fail. This approach results the two similar sequences to result in two different automatons. When we trace the first automaton, from **a** to **b** we have $1:2$, namely input 1 will activate this transition, if the next input is 2. When the transition is taken, we will be at state **b**. From **b** we have two transitions, either to **c** or **d**. Both of them is activated with 2 and requires next input to be 1. When we activate state **a** with input 2, we see that the next input is 3 after the 1, namely $1:3$, but we do not have and such transition from **a**, so for $1, 2, 1, 3$ the transition from state **b** to **a** fails and we achieve 3.22 from 3.21.

However, as seen in Figure 3.21, the automaton can accept $1, 2, 1, 3$ and also can accept $1, 2, 1, 3, 1, 3, 1, 2$. Though, it is not in a global minima yet. There should be a decision about whether keeping the learning at this stage or further learn to get an automaton, that will only accept $1, 2, 1, 3, 1, 3, 1, 2$ but not $1, 2, 1, 3$ sequence at all. This problem will be further inspected in ambigous and exact learning chapters. Namely, the problem is not second order, but in fact the order should be higher.

In this work we further inspect the problem. The problem is not only bound to change from the 1*st*-degree to 2*nd*-degree Markovian processes, but is about having multiple-degrees in a signal. Therefore, we designed on an algorithm that will cope with the multiple degrees of the Markovian processes in a signal sequence.

The proposed algorithm solves this problem, and similar more complex problems and the runtime complexity of our algorithm is less than the proposed complexity.

3.2.3. Two Sequences Mixed in Random - The "OR" Problem

We described learning as going from an ambiguous automaton toward a specific automaton, or a sequence. For most of the cases, the signal is composed of only one sequence. However, it is possible for the signal to be composed of two signal sequences following each other at random rates. In such a case searching for a single sequence can not give the correct result at all. Instead of a sequence, the result should be "or" of two sequences.

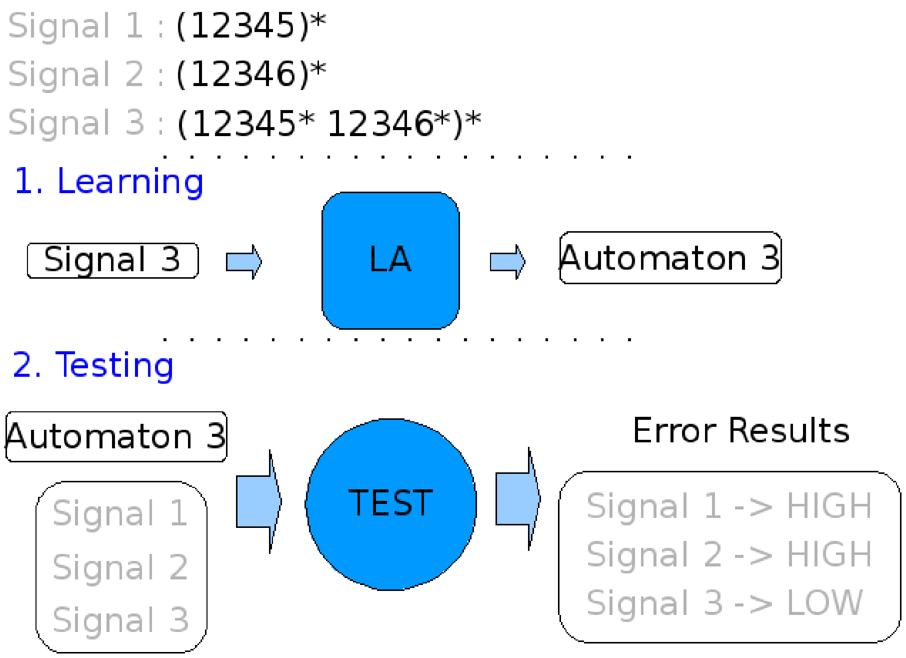


Figure 3.23. A signal may be composed of two different signal sequences.

Figure 3.23 shows two different signal sequences, signal 1 as $(12345)*$ and signal 2 as $(12346)*$ and the generated signal $((12345)*(12346)*)*$ to feed to the algorithm. The algorithm should continue learning until a degree and must stop the learning at an appropriate point. So the resultant automaton should be "or" of those two sequences. So exact learning is not appropriate for this case, because we do not ask for a single, straight path of single tokens. Repetition of tokens or full partial sequences is required in this case, instead.

We used "*" for the repetition of each signal. In this example it is not possible to guess the self recursion count of each signal per the full period. Therefore it is a good choice to keep self loops **intrinsic** and neglecting them until post-processing step. This will help the automaton resist the noise better. The result is more robust but less classifying cabability having automaton, and will help the learning process for such kind of problems.

3.2.4. Affect of Non-Obviousness

In previous sub-sections we mentioned about the affect of the noise, and having more than one signal sequence mixed in random. In each case, searching for the most specific period results in overfitting. However, if the noise ratio is low and the input signal is the result of a single specific period, then we should continue learning until finding the most specific period, that is the straight sequence itself.

The aim of Adaptive Syntactic Pattern Recognition Machine (ASPRM) is searching a more specific automaton from the current automaton. In each step the universe that is represented by the automaton shrinks. If the learning is terminated at some point before reaching the exact solution, then the represented universe will not only include the correct exact period but also some similar periods.

For example, if we have a signal with a period of length 10 and 10% noise is applied on it. This signal may lead an automaton or a straight sequence. If the result is an automaton, then 10% noise will lead 0.1 error on each transition. If we sum the error up, we will achieve error of 1 token per period. Namely the variance of the automaton is 1 token wide. If we provide another source to the automaton with changing only one of the tokens in the period, then the period will result in an error of 1 token per period too and also be accepted by the automaton. However, when the automaton is a straight sequence, then it will only accept the correct sequence as the input, and will reject the noisy parts. So getting a noisy token will result full rejection of all tokens in the next period. One token fails, 100 percent failure is the respond of the sequence. If we provide different sequence to this straight sequence, it will never accept the sequence, so the generated error rate will be quite higher than the same sequence that is used in the learning phase of the automaton, with higher noise values.

So, if the exact sequence is found by the learning algorithm, then the sequence can distinguish very similar periods. However, if the learning algorithm leads to an automaton, and makes the classificiation by looking to the transition probabilities, in high noise rates the classification of the similar periods will be difficult to achieve. *So the classification power mainly comes from the structure of the automaton.*

3.2.5. Constructing Sub-trees (Words)

The signal to be learned may be a result of multiple degree markovian processes. If the correct degree of each part of the signal is not learned correctly, the learning will stay at a level, and it will be impossible to find a more specific automaton.

It is generally the case for markovian processes when the order is higher than first degree. If the order of the markovian process is more than the first degree, and if the algorithm continues updating the probabilities of the transitions and states based on first order, then the learning algorithm will not be able to lead to an automaton that will be a successful classifier for the given input.

In fact, grouping decreases unnecessary searches and by string matching like algorithms it may lead to better automatons, that is also robust to noise.

As stated by the problem statement section, ASPRM should be an algorithm that will search for a more specific automaton from a more general automaton, according to the given input. While searching for the most specific sequence, the effect of noise, not having a specific sequence but having a mixture of sequences, having multiple degree markovian parts in the sequence, and cutting the learning in some degree not to lead to overfitting, should be in mind.

3.3. THE AUTOMATON

The input signal sequence is modelled as an automaton in Adaptive Syntactic Pattern Recognition as seen in Fig. 3.24. Let us start with the definition of each of the components of the automaton.

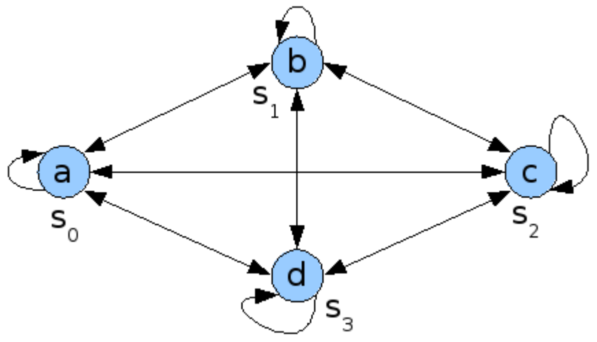


Figure 3.24. The automaton.

3.3.1. State

A state is modelled to keep only one token or a set of other simpler states. As seen in Figure 3.24, s_0 keeps token 'a'. While traversing the automaton we will have a set of active states. When the next signal is taken from the input source, it will be checked with these currently active states. This means, if the token in the next possible states matches with the next token from the input source, then the next state is activated, and placed in the set of next active states.

3.3.2. Token

ASPRM can be used for various kinds of problems, from natural language processing to multi-dimensional signal processing operations. Therefore we abstracted the basic unit of signal as a Token. It enables the system to work independent of the type of the pattern. After preprocessing, either a multi-dimensional signal or a character will be a token.

3.3.3. Automaton

An automaton is a state map that is composed of states and connections between them. In this implementation an automaton is modelled as a self-constructive layered structure. The top layer contains options, transitions and the leaves contain the tokens.

3.3.4. Option

Option is a kind of state that keeps set of simpler states. It is modelled as a tree structure. Its states may be either tokens, or the other states. When in the input we see a sequence very frequently, and it makes the system sure about its stable ordering, the signals composing this sequence are combined together to construct an option. Using options makes the Markov decision process a semi markov decision process. It increases the granularity of learning. Instead of looking for matching of alphabet elements with state informations, algorithm looks for activation of options, activation of sub-groupings, sub-sequences.

Using options makes the automaton simpler, and decreases the complexity exponentially.

3.3.5. Pattern

Input data may be multi-dimensional, and may have different properties. The input is taken as a sequence of patterns. An ECG signal, speech data, or OCR data may be the input. Therefore, the system is designed not to be dependent on the type of the signal source. For a better learning, the features should be extracted carefully.

Different patterns may result to the same token. For example, “a“ character maybe written differenly, however its meaning in a word is always “a”. So, for every pattern representing “a” character, we will use token “a”, with some error. This error can be handled by the algorithm, however using a_1 and a_2 interchangeable for token "a" will disturb the interrelationships, and stability can not be achieved in this situation.

If period is "fuat", and if we get "fu$a_1|a_2$t", then exactly learning the sequence, as a straight sequence, is impossible. If it is valid for more tokens, then we will see something like "$f_1|f_2u_1|u_2a_1|a_2t_1|t_2$", and it is quite harder to find the relationships in such a case. Namely, in such a situation, we should remain at markov level of one, otherwise we would have 16 different sequences, but which had to be the same.

3.4. TRAVERSAL

ASPRM uses the automaton to model the input sequence, and tries to find a better model for the given input sequence. It collects statistics about the probability of correctness of each state and transition. This probabilities are then used to make structural modifications on the automaton.

While learning, the algorithm keeps an automaton, a set of active states, and for statistical purposes current probabilities of each transition and state. As new input token arrives, a set of next possible states are checked. The automaton decides about the next possible active states. If a state has an input transition from a currently active state, then it will be added to the next possible active state set. The next possible active states will be compared with the input, and according to the ratio of matching they will become the current active state of the next step.

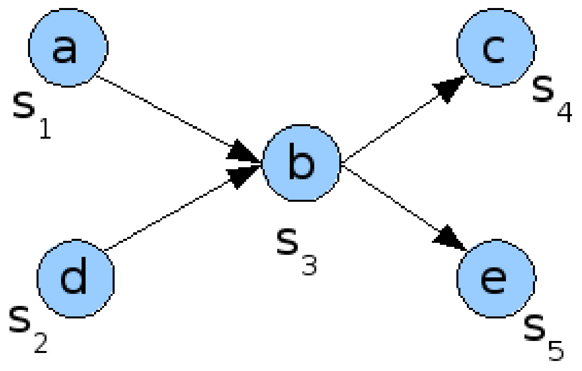


Figure 3.25. A part of an automaton. Active state, activating states, and next possible active states.

For example look at Figure 3.25. Say s_3 is the currently active state, and it was activated by state s_1. The next input is taken from the input source and compared with the token values in the next possible states, which are s_4 and s_5. If the input is "c", then s_4 is activated. So we read the traversal as, active state s_3 was activated by state s_1 and activates state s_4. According to this information algorithm updates the probabilities, and at the end of an epoch structural modifications take place.

In comparison of the next possible active states with the input token we apply some matching algorithms like radial basis activation function, and euclidian distance function.

The transition between currently active state and newly accepted active state is the transition which is traversed. The probability of the transitions and the states leading to new active states should be increased, and others should be decreased.

There are different traversal schemes, those have various activation, traversal and rewarding methodologies.

3.4.1. Basic Traversal

A traversal is basically moving from a state to another according to an input token. Each movement will lead a statistical information. Reinforcement learning principles are applied to update the probabilities of the transitions and the states. If a state is activated, the state which led tothis state to be activated and the transition between them is rewarded, while other states and transitions are punished.

Lets again look at the Figure 3.25. The state s_3 was the active state, which was activated by s_1 and activating the state s_4. So the interpretation of this movement for updating the probabilities is so;

- Because s_4 is activated, increase the probability of transition from s_3 to s_4
- Because s_5 is not activated, decrease the probability of transition between s_3 to s_5
- Because s_3 makes a successful traversal, update the probability of transition between the activating state s_1 and s_3
- and decrease the probability of transition between s_2 and s_3.

$$
\begin{aligned}
P_{ts_3,s_4} &= P_{ts_3,s_4} + \alpha(1 - P_{ts_3,s_4}) && (3.3)\\
P_{ts_3,s_5} &= P_{ts_3,s_5} + \beta(0 - P_{ts_3,s_5}) && (3.4)\\
P_{s_3} &= P_{s_3} + \delta(1 - P_{s_3}) && (3.5)\\
P_{ts_1,s_3} &= P_{ts_1,s_3} + \omega(1 - P_{ts_1,s_3}) && (3.6)\\
P_{ts_2,s_3} &= P_{ts_2,s_3} + \sigma(0 - P_{ts_2,s_3}) && (3.7)\\
P_{s_1} &= P_{s_1} + \eta(1 - P_{s_1}) && (3.8)\\
P_{s_2} &= P_{s_2} + \zeta(0 - P_{s_2}) && (3.9)
\end{aligned}
$$

In this way we increased the traversed path's probability, and decreased the probability of the other possible ways.

3.4.2. Intra-Word Statistics

As mentioned earlier, an automaton is composed of states and transitions. A state is like a tree of inner states. If the traversal does not lead to a pass from one top level state to the other, then it is a movement in the leaves of the tree of a state. This kind of movements is called intra-word traversal.

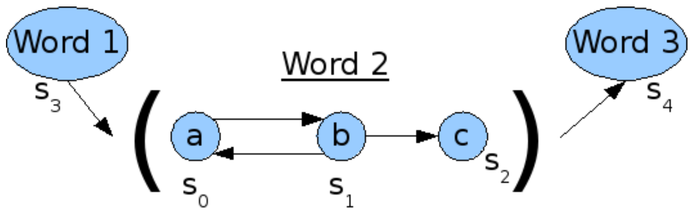


Figure 3.26. A part of an automaton, includes an inner automaton, a word

As we look at Figure 3.26, $Word_2$ is an inner automaton, which we name a word. Movements between s_1, s_2 and s_3 will be inner traversal, intra-word traversal. The movements into or out of the word is inter-word traversal.

Intra-word statistics are collected to be used in the post-processing phase to estimate the powers of the self-loops.

3.4.3. Inter-Word Statistics

If traversal leads one of the top level active state to change to the another, then this kind of traversal is called inter-word traversal. The statistics collected by the inter-word traversals, are used to change the structure of the automaton. The intra word movements do not affect the statistics, because we have already constructed them because of previous statistical information. We accepted them as valid words.

3.5. STRUCTURE MODIFICATIONS

In our algorithm learning to mainly the changes on the structure of the automaton. At the start, the automaton will accept any sequence as the correct sequence. Later, as the learning continues, some periods will remain acceptable, but some others will get out of the tightened boundaries. Various structure modification methods are used in the algorithm like removal, splitting, and grouping.

As mentioned earlier, the learning process is like a search for a global minima. The removal and the grouping steps are like exploitation, which decreases the entropy and makes the automaton more stable. However, the splitting phase is like exploration, which reforms the automaton and represents the same automaton with more states. Split operation increases the entropy of the automaton and makes the automaton to jump to yet another search space. The search continues in this new space as looking for the minima of this new space.

3.5.1. The Algorithm in Brief

Let's first look at the brief description of the algorithm. In next subsections each operation is detailed by an example.

1. Start with a fully connected graph of alphabet tokens
2. With input sequence traverse the automaton and collect statistics about usage of transitions and states
3. Check for removal
   (a) Remove the transitions with probability less than a threshold
   (b) Remove the states with probability less than a threshold
   (c) Remove states with no incomming or no outgoing transition
   (d) Remove transitions of removed states
   (e) Continue with (c) until no removal

    (f) If no state remains finish with failure
    (g) If more than one automaton is generated then finish with failure
    (h) If any removed, then continue with (2)

4. Check for grouping states to achieve options
    (a) Find states with single incomming transition, S_{si}
    (b) Find states with single outgoing transition, S_{so}
    (c) Find a path
        i. If a state in S_{si} has a transition from a state in S_{so} they are in the same path
        ii. Add all such states to one option, which is both in S_{si}, and S_{so}
        iii. Head of the option will be the state in the S_{so}, but not in the S_{si}
        iv. Tail of the option will be the state in the S_{si}, but not in the S_{so}
        v. If cycle occurs finish the algorithm as success
    (d) Construct the option
        i. Transitions to the head of the path will be incomming transitions of the option
        ii. Transitions from the tail of the path will be outgoing transions of the option
    (e) If generated any option, then continue with (2)

5. Check for bias elimination
    (a) Find the state whose entropy is maximum, namely probability values at the transitions is well distributed, and the state also has high probability value
    (b) Generate as many states as the outgoing transitions of the selected state
    (c) For each generated state connect all input transitions to them and connect only one outgoing transition per each
    (d) Continue with (2)

3.5.2. Start of the Algorithm

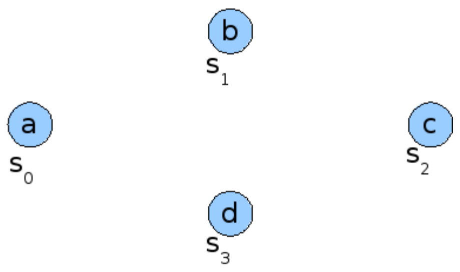


Figure 3.27. Alphabet consist of four patterns

Initially, an alphabet of the input data will be generated. As seen in Fig. 3.27, we have four tokens ’a’, ’b’, ’c’, and ’d’ in the alphabet. The period of the input data used is “cabcad” in this example. Our aim is finding this period with structure modifications. Iteratively this period will be fed to the algorithm with some noise addition.

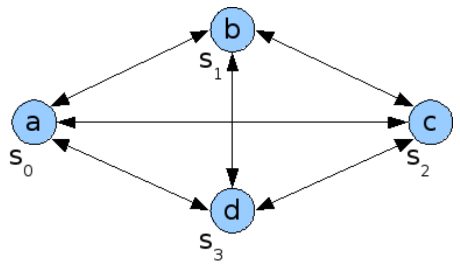


Figure 3.28. Initial automaton, all of the states are connected to each other

For each token we use a state and the learning process starts with a fully connected graph of these states as seen in Figure 3.28. In a fully connected graph, each alphabet element can be the next input that is acceptable. So it represents the universe of all possible sequences. Our aim is decreasing the volume of the universe to accept only more specific sequences. In fact, it is not obligatory to start with a fully connected graph. It is for the case of knowing nothing about the signal, because it encapsulates all the cases. If we have some prior information about the signal, for example prior transition probabilities between two tokens, we can use this prior information to generate the initial automaton.

3.5.3. Removal

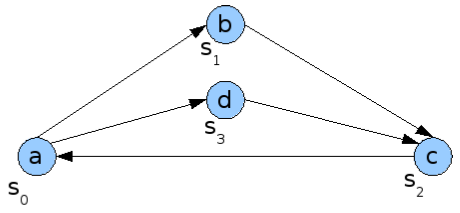


Figure 3.29. The automaton after the first removal operation

To move from generic universes to specific universes the algorithm utilizes some structure modification operations. With given data, the current automaton is traversed and statistical information about each state and transition is collected. The transitions having less probability than a threshold value are deleted from the automaton. As seen the transition from state s_3 to state s_0 is removed. Because in the input sequence "da" sequences could not be matched more than a threshold value to save the transition from removal. The states, containing no incomming, or no outgoing transition, are also deleted. Each removal operation results in the new automaton not to represent some sequences any more. Therefore removal operation is a movement toward the correct, possibly unique, sequence.

As seen in Fig. 3.29, the first removal operation resulted in a much cleaner automaton. However, this automaton still contains states which have more than one option as the transition to take. The entropy of the first automaton decreased by the removal operation, however the searched sequence may be more obvious than this automaton's representation.

3.5.4. Grouping

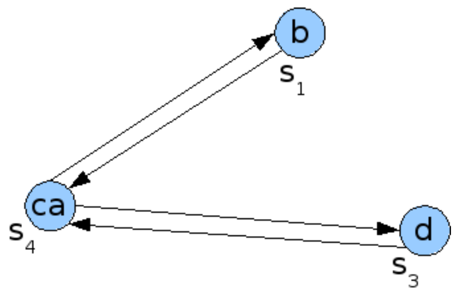


Figure 3.30. Two states are grouped, and an option “ca" is constructed

If we inspect the Fig. 3.29, we can see that after s_2 there is only one option which is s_0 as the state. So the knowledge of getting “a” as the next alphabet element after “c” is extracted.

In order to make the automaton learn data, that is the result of a system operating as multiple-degree markovian process, grouping operation is devised. So, the states are grouped in a logic to achieve some options. In Figure 3.30, s_2 and s_1 is grouped, so the option s_4, “ca”, is achieved. The incoming transitions of state s_2 will be the input transitions of this new state, and the outgoing transitions of state s_1 will be the outgoing transitions of it. From now on the “ca” will be thought as a single pattern, and it will be activated if and only if both ’c’ and ’a’ comes consequently. This increases the degree of markovness for this option from one to two. As a result, the resultant automaton has states, some in the first order markov, and some in the second order markov level.

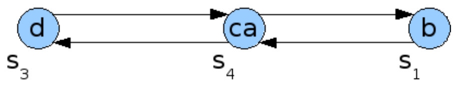


Figure 3.31. Resultant automaton after the first grouping operation

When we look at the Fig. 3.31 the results of the learning can be seen better. The learning evolves the automaton to such a simple shape. However, this shape also represents a huge universe, and the universe may be narrowed more.

3.5.5. Split

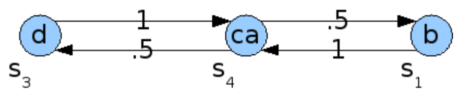


Figure 3.32. Probabilities of each transition

Fig.3.32 shows the probability values of each transition after another traversal done on the automaton. The decrease in the entropy means stability that means going toward the correct sequence. In learning we can focus on the source of the entropy or the source of the maximum error. There are two methods one uses the entropy values and the other uses the error, in changing the shape of the automaton to the same automaton with cleaner paths.

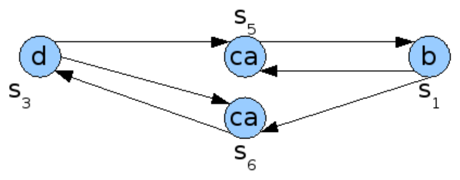


Figure 3.33. Resultant automaton after the state with the highest entropy is split

As seen in Fig.3.33, the state which is the source of the highest entropy, s_4, is split into two states, s_5 and s_6. The resultant automaton represents the same universe as the first one. However the outgoing transitions are split, and the entropy in s_4 is eliminated. In this new automaton view, the source of each outgoing transition of s_4 is known. So if "d" cames after the "ca" it will be known that it will be the state s_6, that should be the active node.

As seen, these two automatons represent the same universe. However, the second automaton provides more information about the flow, and represents the universe better.

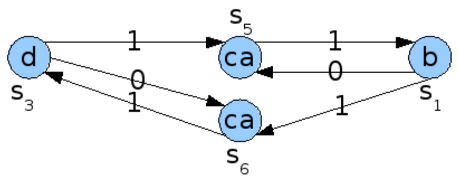


Figure 3.34. Statistics of current automaton for the first order markov data, “d c a b c a".

If the period is “dcabca”, then the probability values of the transition will be as shown in the Fig. 3.34. So the split of the “ca” lead us to reveal two unused transitions. In fact, the split increases the entropy, however it leads in the next steps, finding better automatons, which are more specific.

In reinforcement learning we have exploitation and exploration. Removal and grouping operations can be seen as exploitation, those decrease the entropy, and splitting can be seen as exploration, which increases entropy but leads to better automatons in the next steps.

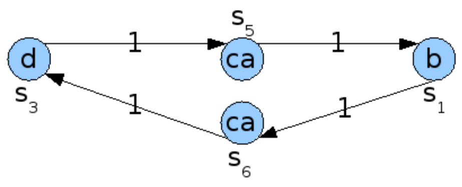


Figure 3.35. The most specific automaton for the first order input

After traversal we found some transitions having probability less than the threshold value. So when we remove them the automaton in Fig. 3.35 is achieved.

3.5.6. Different Periods to the Same Automaton with Different Transition Probabilities

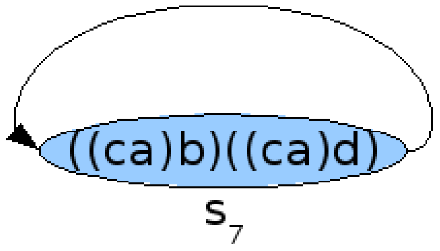


Figure 3.36. The most specific automaton for the first order input

When the next grouping operation is performed the resultant automaton "((ac)d)((ac)b)" is achieved.

We made the self cycles as intrinsic to each node, because of the fact that self cycles make the automaton more noise flimsy. The pharanthesis in the final automaton represents the sub-automatons, which have self transitions. If we know the ratio of the noise to be low, we may also contribute the self recursions as transitions and let them to be removed too. In post processing we searched for some methods to overcome the adverse affects of self recursion assumption.

If we include the self recursion as transitions in the automaton, the resultant automaton would be a straight string. In the current result we differentiate two periods "acdacb" from "acdacdacbacb" by the probabilities of the self transitions. However, if the self recursions were also thought as transitions, the resultant automaton for "acdacdacbacb" would be different than the resultant automaton of "acdacb" sequence.

If the learning process continues, the algorithm may become capable of differentiating these two periods too.

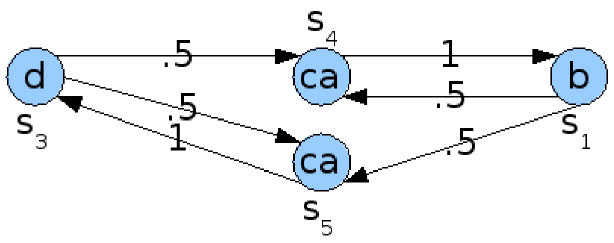


Figure 3.37. Probabilities of the transitions after second split for "abcabcabdabd" case

As seen in Fig.3.37, if the period was "abcabcabdabd", the transitions from s_1 to s_5 and from s_3 to s_6 will be still alive. So, the learning method can differentiate these two periods with the help of probabilities.

However, to make the automaton more robust against the noise, we made a generalization, that is, assuming self recursions for each node, or option. That leads to two different periods to be seen as the same automaton structure.

In fact, our aim is not to depend on the probability values on the edges, but to find an automaton, that itself represents the period of the signal.

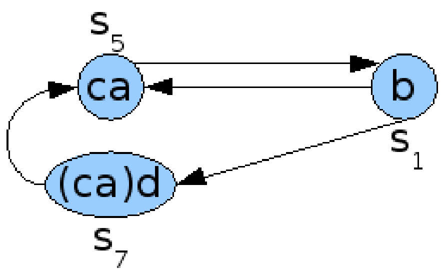


Figure 3.38. The result of the first step of the second grouping operation.

As seen in the Fig. 3.37, there is only one possiblity for the outgoing transition of s_6, that is s_3. So the algorithm combines these two states and constructs the option "(ca)d", as seen in Fig.3.38.

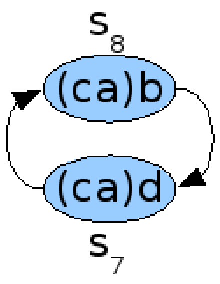


Figure 3.39. The result of the second step of the second grouping operation.

As seen in Fig.3.39, the algorithm should also connect s_1 with s_5. So s_7, "(ca)b", is constructed.

The resultant options can be combined too. So, the algorithm generates the automaton "((ca)d)((ca)b)". Again the same result is achieved in the intrinsic self recursion case. However at this time inner transition probabilities of s_7 are different than the result of the first learning.

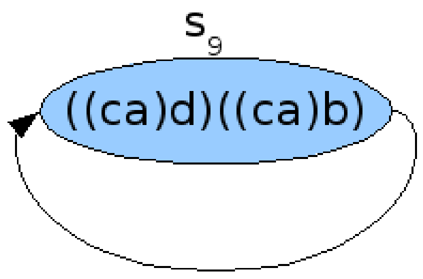


Figure 3.40. First step of the second grouping

3.5.7. Exact Learning and Effect of Noise

We do not want the probabilities to show the difference between two automatons. We want two automatons to represent different universes without the help of their transition probabilities. However, this will require each data to have exactly one straight period.

Generating options will result in the automaton to have edges for different markov-degrees. However, longer the options, more prone to bad effects of the noise.

Generally it is not usual to have a straight period in the data. Therefore, having self recursions in the automaton, and limiting the size of the generated options are good decisions. They limit the automaton from being more specific, but make it more robust.

To show the capabilities of the learning algorithm we also implemented another version that keeps the self recursions as transitions, and showed the results for sequences "abcabd" and "abcabcabdabd". They resulted in two different automatons. In fact in the older case, we were separating two of them via looking at the probabilities. However, in this new case the two automatons, are also structurally different from each other.

3.6. EXACT LEARNING

In previous two sections we assumed each state to have a self loop. This intrinsic self loop makes the automaton more persistent to the noise. Furthermore, in some examples there is no single sequence a signal can be represented with. However, there may be examples that the signal is represented by a single, and quite straight sequence. Therefore, this exact learning approach is implemented to achieve a full learning session, searching for a straight sequence.

In this section we will describe the algorithm as having no intrinsic self recursions on the states. So instead of applying post processing to guess the power of the states, this algorithm will both learn the structure of the automaton, and the power of each state together.

In the first section of this chapter we mentioned about the affects of the non-obviousness. As the noise ratio increases, if the result of the learning is not a single sequence but an automaton, then it will be possible to have some periods similar to the correct period to get less error rates than the noise added original signal. However, if a single correct sequence is found, no matter how similar the test input's period is to the original signal, the resultant structure will never accept the similar period as the original period.

Having an exact solution will decrease the robustness to the noise in learning phase, but the learned automaton will be a strict classifier. While learning and testing instead of using intrinsic self loops, other techniques according to the problem at the hand can be implemented. String matching, fuzzy matching techniques can be some of the group matching, robustness enhancing techniques.

So, let us describe the differencies of the exact sequence learning from the previous algorithm.

3.6.1. Exact Automaton

Instead of having an intrinsic self loop, states should keep one more transition to themselves. This transition may be removed later, or may result to splits if there are more than one power value in the period for a state.

In the previous non-fully obvious version of the automaton learning, we neglected the power of the states, so one state might be visited more than once in the final automaton. However, in the exact learning, the result of the learning is a single sequence, that should exactly match to the period to be correct.

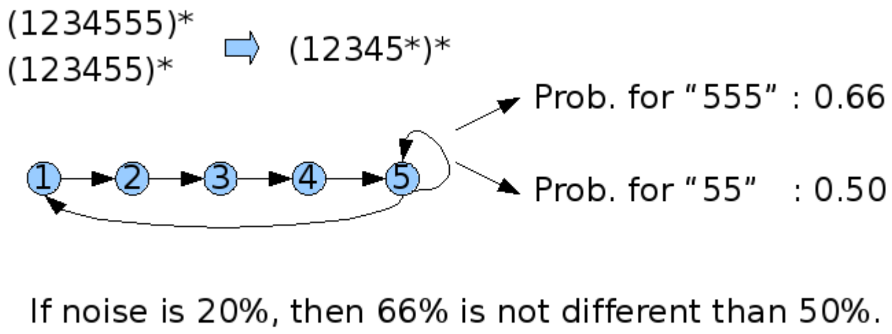


Figure 3.41. Using the probabilities as the classification criteria may not be always possible.

As seen in Fig. 3.41, in the previous implementation the two periods "(1234555) ∗" and "(123455) ∗" will produce the same automaton with different probability values for the "5"'s. Neglecting the power of the tokens will result automatons with same structures but different transition probabilities.

While learning, it is a good behaviour to keep the automaton as "(12345∗) ∗", so the affect of the noise in the power of the "5" will not disturb the learning very much. However, if the purpose is also separating these two periods, namely if the powers are also important, the transition probabilities may not be enough.

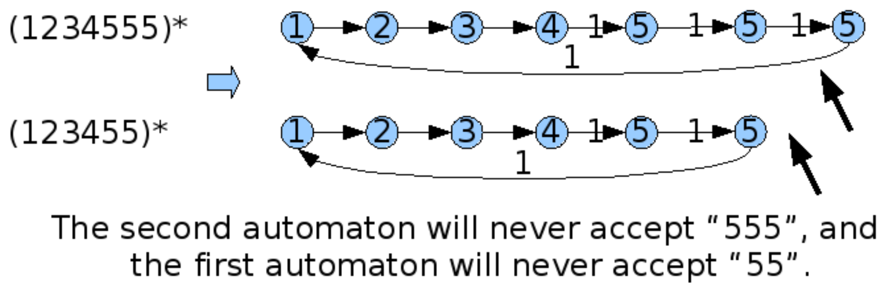

(1234555)*
(123455)*
1
1
1
1
1
1
1
The second automaton will never accept "555", and the first automaton will never accept "55".

Figure 3.42. Exact automaton is a full classifier, that can also classify too similar periods.

As already mentioned on previous sections, classifying signals with only looking at the probabilities will be not enough for some cases. This is one of those cases. For just a noise level of 10 percent, the original signal's error rate will be more than the second period's error rate on the learned period as seen in Fig. 3.41. However, if the exact result is found, the result is like the Fig. 3.42.

3.6.2. Start of the Algorithm

The exact algorithm is not much different than the original algorithm. Only the states will not have self loops any more. So the processing of the states, and structure modification algorithms are adapted for the case.

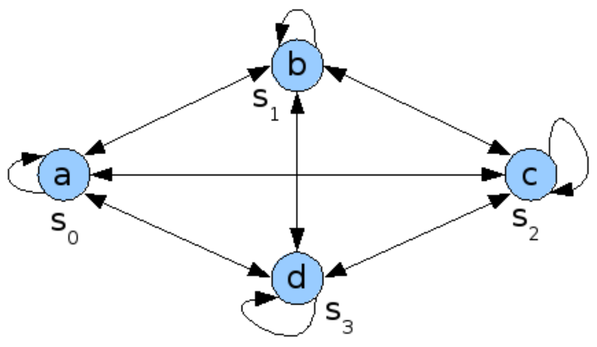


Figure 3.43. The initial automaton for exact learning.

The only important difference is at the starting automaton. Because in the previous non-exact version of the algorithm the self loop was intrinsic, and it is not intrinsic in this version, the self loops must be placed as external transitions. Later in the learning phase those self loops can be removed, grouped etc. and can result different power values for each token as a structure in the automaton. However, in the previous algorithm the power was just a probability and having no self loop was just like having nearly zero probability for the self loop intrinsic parameter.

Figure 3.43 shows the start of the algorithm. In the non-exact learning version of the algorithm, when the algorithm is tested with "(abcabd)*" and "(abcabdabdabc)*", and the same automaton with a little difference in the probability of the self loop is achieved. This probability difference is used in finding the correct power at the post-processing phase. If the signal is quite noisy it is difficult to find the correct power values.

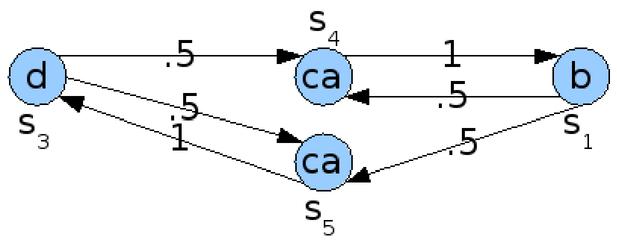


Figure 3.44. Exact learning will produce different automaton than non-exact learning.

So, let us continue with the automaton shown in Fig. 3.44, and continue learning as exact case.

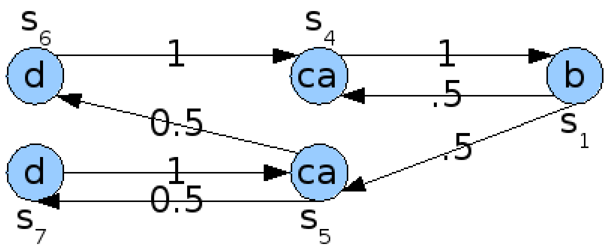


Figure 3.45. The automaton as the state with the token “d" is splited.

As seen in Fig. 3.44, the entropy is high at the state with token “d”. So the state is splited into two states. The entropy is moved to the state at the bottom with tokens, “ca”. As seen in Fig. 3.45, the output transitions of the states with token “d” shows probability of 1.

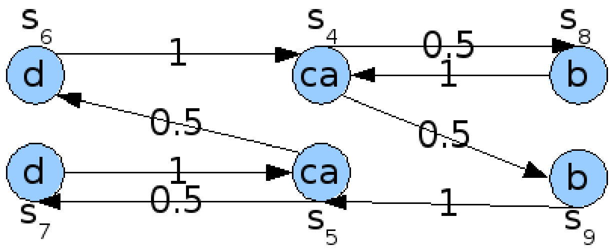


Figure 3.46. The automaton as the state with the token “b" is splited.

The newly achieved automaton is still stable, no removal or grouping can be taken either. So the source of maximum entropy is searched and found to be the state with token “b”, and it is splited as seen in Fig. 3.46.

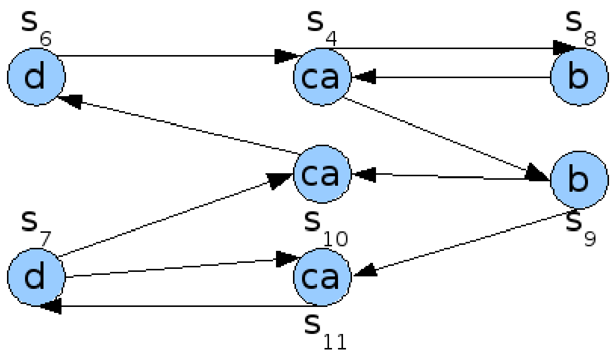


Figure 3.47. The automaton as the bottom state with tokens “ca" is splited.

The newly generated automaton is still stable, and again the biggest source of the entropy is found and splited. It is the bottom state with “ca” tokens, and the new automaton is like in the Fig. 3.47.

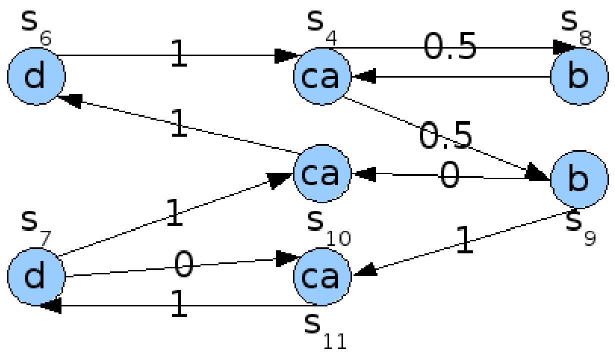


Figure 3.48. Statistics for the automaton when the results of the actions are propagated two step back.

So after enough exploration yet another universe is achieved in which some transitions are not required. Statistics in Fig. 3.48 shows that some transitions’ probabilities are below the removal threshold.

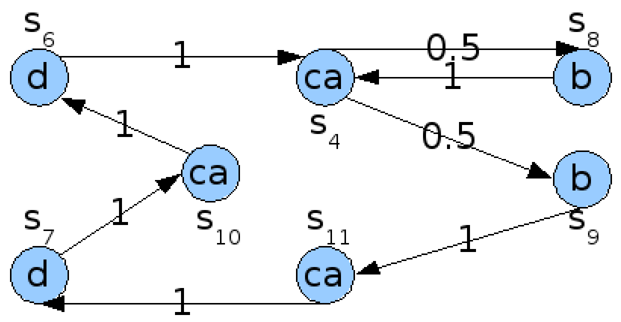


Figure 3.49. The resultant automaton of the exact learning is different than the result of non-exact learning.

After the removal of the noisy transitions the automaton in the Fig. 3.49 is achieved. It is important to note that, in the Fig. 3.37 the automaton was already stable, and could be used as a classifier by using the transition probabilities on the transitions. However, as seen, the automaton was not the final, global minimum, and we reached a better automaton can be found. This shows the power and correctness of our algorithm in elimination of the non-obviousness.

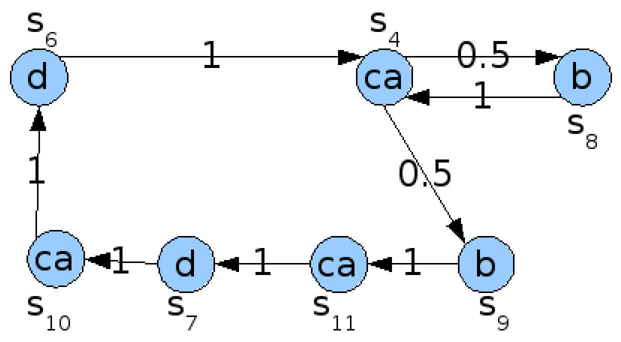


Figure 3.50. The automaton has a long straight arm.

After this removal for some states there remained only one option to go to the next state. Algorithm groups these states. In fact grouping is not all required in exact case, but it helps learning. When states are grouped, we force the input to be exactly the same sequence as the unique opportunities in the automaton. Namely, if the automaton shows one state to come after another always, then grouping of them will make them a single pattern. If this pattern is parsed from the input than this state is activated, but if this pattern is not parsed in whole then this state is not activated. In fact this also makes the automaton more prone to bad affects of noise.

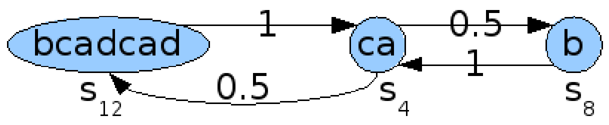


Figure 3.51. The automaton, when the states are grouped. A cleaner view is achieved.

Grouping has two effects. One is increase of sensitivity to the noise, and the other is generation of sub-automatons. If the underlying signal sequence will be better represented by sub-automatons then grouping some states will give better results. However if it is not the case, grouping will only increase the effect of noise. In non-exact learning, grouping will lead generation of new complex alphabets, and this will give capability of making better matches like string matching, fuzzy matching etc. So the effect of the noise is alleviated and a better learner, that generalizes the signal better, is achieved.

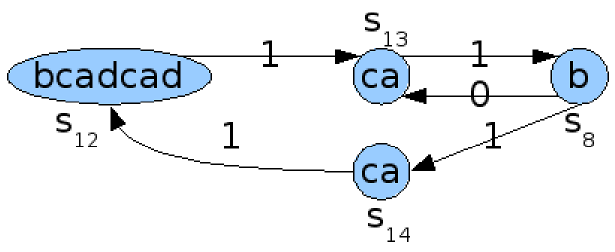


Figure 3.52. The automaton when the state with tokens “ca" is splited.

The last source of high entropy is also splited into two. The resultant automaton has no entropy, it is stable and it is the global minima, not the local minima.

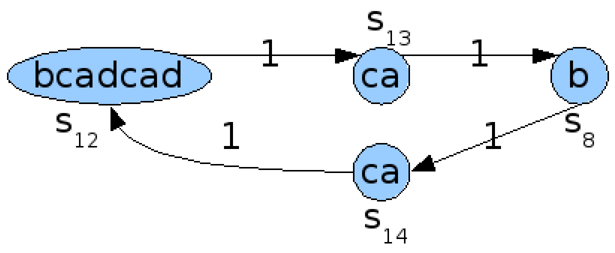


Figure 3.53. The automaton when the transition whith is the last source of the entropy is removed.

3.6.3. Result of Exact Learning

After the removal of the last non-used transition no transition remains non-used or to be searched in other universes. So learning finishes with this last grouping.

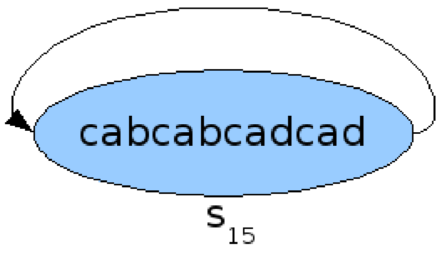


Figure 3.54. Exact learning generated a better automaton.

As seen in Fig. 3.54, the resultant automaton of exact learning is different than the result of non-exact learning shown in Fig. 3.40. The automaton can separate inputs “(abcabd)*“ from “(abcabdabdabc)*” easily for high noise ratios, however the first automaton can make this separation for only low noise rates.

In high noise, and if the input a mixture of more than one signal sequence, searching for a specific sequence will only lead overfitting, and removal of some part of the period, that is not noise. So in some cases it is better not make an exact learning but an acceptable noisy ambiguous learning.

So according to the problem, the level of noise to neglect, the degree of exactness should be chosed. Either a fuzzy string matching algorithm, or different preprocessing algorithms may lead better results for different problems.

3.7. CALCULATION OF THE ERROR RATES

When the ASPRM finishes its learning the input sequence, its result will be an automaton, and the probabilities of the states and the transitions on the automaton. According to the aim of usage this probabilities and the automaton will be the base for calculations.

In the previous work [5], a tracker algorithm is implemented. This algorithm tries to find all possible power values for the transitions. So, it checks the possible sequences with the input data, and tries to find the most possible sequence. The most possible sequence is declared as the found period. When the found period is checked with the original period, the rate of success is calculated.

Tracker algorithm is quite powerful. It tries to find possible periods from the universe of the current automaton. If the learning continues, then the automaton will search for better minima positions, and the universe will shrink. In the environments with high noise, this will make the learning more difficult. However, if the tracker algorithm is applied, the learning can be stopped at some level and all possible periods can be checked.

In fact, the aim of the learning is elimination of the possible periods by learning, not by brute-force. Furthermore, when there is no exact period, the tracker algorithm will not be usable at all.

So two different methodologies are proposed. When the data is known to have exactly single period, and the level of noise is low, then exact learning should be used. It learns the data until reaching to the global minima, namely a single sequence. The other methodology is testing the classification power of the learned automaton. While the learning continues, the automaton must be better and better in classifying the data similar to the original one, from the others those are not very similar.

3.7.1. Exact Learning

Exact learning mode of the ASRPM finds a single sequence. Test of the correctness of the found period is quite straight. Check whether the original period is equal to the found period. Because each transition has probability of one, and each state has nearly equal probabilities, when the noisy input signal is given to the automaton, the resultant error rate will show the amount of noise in the input signal. So the resultant automaton can be used to find the noise amount of the signal too.

3.7.2. Classification

In addition to the found automaton and the probabilities, there should be a formula that will calculate the similarity of the given signal sequences to the results. This formula must give low error rates for the similar signal sequences and high error rates to dissimilar ones to the input signal.

The signal data to be tested is passed to the automaton, and new probability values and number of cuts in the traversals are recorded. The difference between the original probabilities and number of cuts will be used to calculate the error rate. Bigger the difference in the probabilities, higher the error rate. Bigger the count of cuts, higher the error rate.

Let Pt_{ij} be the probability of the transition from state **i** and to state **j**, $Pt^{'}_{ij}$ is the probability of the transition for the test data, and **N** is the number of transitions.

$$Error = \frac{\sum_1^N Pt_{ij} - Pt^{'}_{ij}}{N} \tag{3.10}$$

Equation 3.10 shows the formula for the probability difference between learned automaton and the test data.

Likewise, the difference between the probabilities of the states and number of cuts will be calculated, and their average will be the error rate. In the next section we will show the affect of the learning on the error rate.

4. TESTS and RESULTS

In the second section we stated various problems to be solved by the designed algorithm. In this section, the learning algorithm will be tested for its adequency in solving these problems.

First we will check the structure modification operations, whether or not each operation leads to the intended results. Next, effect the of noise in the learning will be inspected. In the third test, the ability of the algorithm in learning multiple-degree markovian processes will be tested. Next, the algorithm will be tested for the input that is composed of random mix of two periods. In the last test we will assess the automaton as a classifier. Similar periods will be generated with the help of genetic algorithms, and the resultant automaton of the learning algorithm will be used as the classifier to classify the input sequence from the similar periods.

4.1. TESTING FOR CONTINUOUS LEARNING

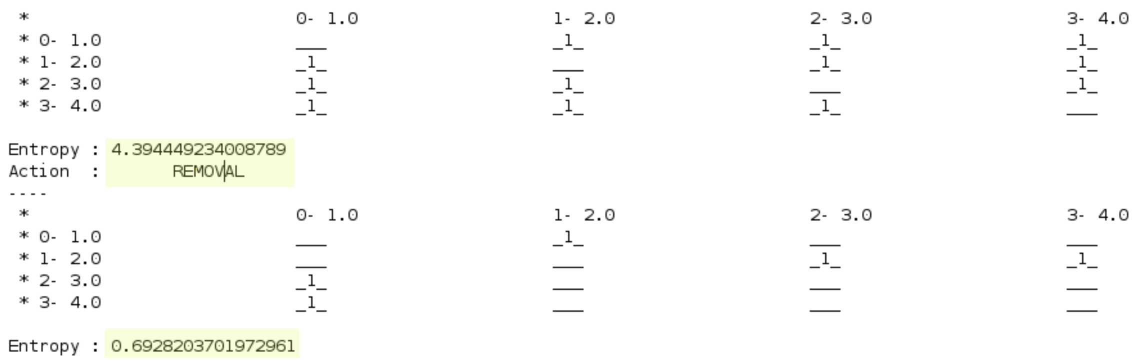
While describing the steps of the algorithm, we stated the algorithm resembling search algorithms. In the section the structure modification operations and their intended results are discussed. Next we will make a sample learning and inspect each structure modification operation.

We have three structure modification operations, and one more operation for learning the statistics of the states and the transitions. We start with an automaton that accepts every possible sequence that the given alphabet can generate. So the operations will perform a search from this universe toward the global minima automaton.

The entropy value means the variance of the automaton. If the entropy of an automaton is high than it means the range of acceptance of the automaton is high too.

So, the operations decreasing the entropy are learning operations that leads automaton to the nearest minima. However, split operation will carry automaton to different spaces, and this will increase the entropy. So, split operation saves the automaton from local minima.

4.1.1. Removal



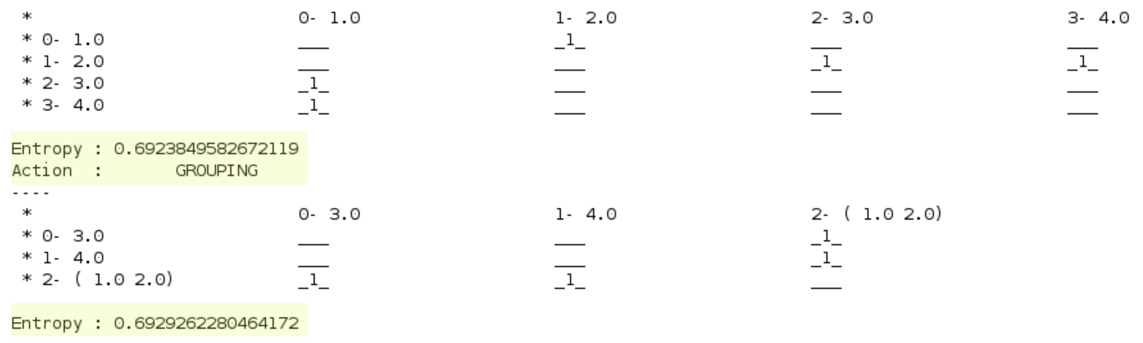
```
 *                    0- 1.0          1- 2.0          2- 3.0          3- 4.0
 * 0- 1.0             ___             _1_             _1_             _1_
 * 1- 2.0             _1_             ___             _1_             _1_
 * 2- 3.0             _1_             _1_             ___             _1_
 * 3- 4.0             _1_             _1_             _1_             ___

Entropy : 4.394449234008789
Action  :       REMOVAL
----
 *                    0- 1.0          1- 2.0          2- 3.0          3- 4.0
 * 0- 1.0             ___             _1_             ___             ___
 * 1- 2.0             ___             ___             _1_             _1_
 * 2- 3.0             _1_             ___             ___             ___
 * 3- 4.0             _1_             ___             ___             ___

Entropy : 0.6928203701972961
```

Figure 4.1. The initial automaton, and change of the entropy after the first removal.

First operation is removal. Figure 4.1 shows the effect of removal on the automaton and on the entropy. Each removal operation will lead some transitions or states to be removed, so the space of accepted periods will be shrinked by this operation. This operation decreases entropy and leads the automaton to reach the nearest minimum, and stable shape. Figure 4.1 shows that, the initial automaton has entropy of 3.39 when the probability of each transition is equal. When the first removal operation is applied over the automaton, the entropy drops to 0.69.

Next operation is grouping. Grouping operation does not have any effect on entropy. In fact, in grouping we are combining states, those have no other chance than coming after one other. So the probability of the transition between them is already 1.0, and the addition to the entropy is already 0.0. Combining those states will not lead any entropy change. The main effect of grouping comes in traversal. When states are combined, the whole combination is activated or whole is failed, which leads multiple-degree markovian processes to be learned successfully.

4.1.2. Grouping



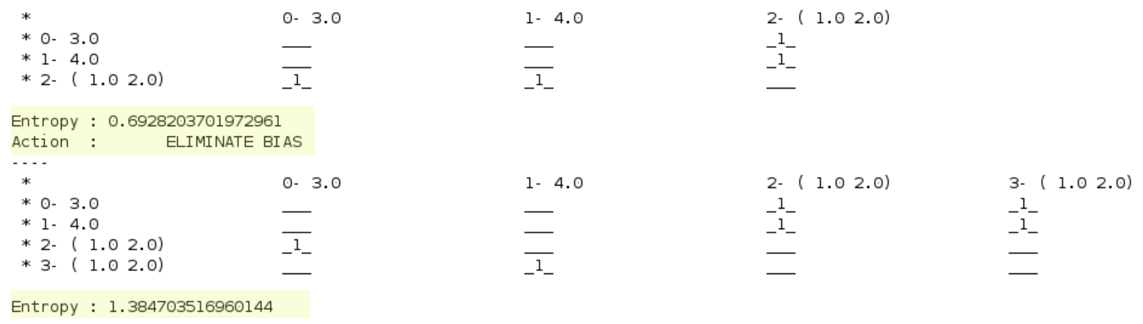
```
 *                    0- 1.0        1- 2.0        2- 3.0        3- 4.0
 * 0- 1.0             ___           _1_           ___           ___
 * 1- 2.0             ___           ___           _1_           _1_
 * 2- 3.0             _1_           ___           ___           ___
 * 3- 4.0             _1_           ___           ___           ___

Entropy : 0.6923849582672119
Action  :       GROUPING
----
 *                    0- 3.0        1- 4.0        2- ( 1.0 2.0)
 * 0- 3.0             ___           ___           _1_
 * 1- 4.0             ___           ___           _1_
 * 2- ( 1.0 2.0)      _1_           _1_           ___

Entropy : 0.6929262280464172
```

Figure 4.2. The automaton and the entropy after the first grouping.

Figure 4.2 shows the effect of the grouping on the automaton and on the entropy. As stated, the entropy value did not change. There is only one transition emerging from state with token “1”, and it enters to state with token “2”. The state with token “2” has also only one incomming transition. So these two states are combined. Algorithms, like fuzzy string matching, can be applied for checking the grouped states with the input sequences.

The final operation is the split operation. This operation saves the automaton from local minima, and must lead the entropy to grow initially, and decrease much more later.

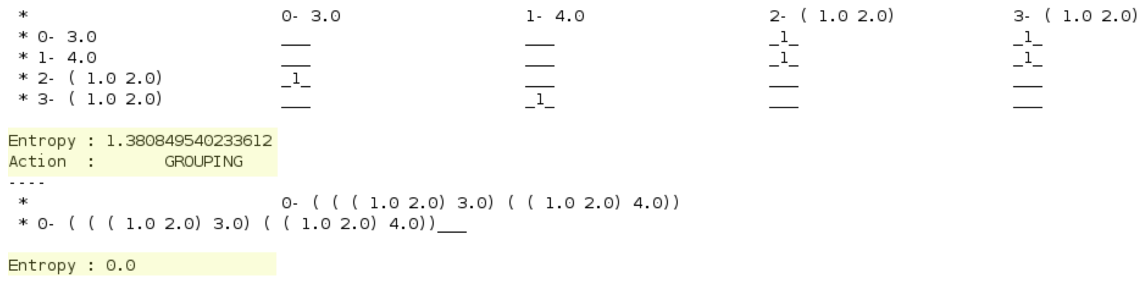
4.1.3. Split



```
 *                    0- 3.0        1- 4.0        2- ( 1.0 2.0)
 * 0- 3.0             ___           ___           _1_
 * 1- 4.0             ___           ___           _1_
 * 2- ( 1.0 2.0)      _1_           _1_           ___

Entropy : 0.6928203701972961
Action  :       ELIMINATE BIAS
----
 *                    0- 3.0        1- 4.0        2- ( 1.0 2.0)   3- ( 1.0 2.0)
 * 0- 3.0             ___           ___           _1_             _1_
 * 1- 4.0             ___           ___           _1_             _1_
 * 2- ( 1.0 2.0)      _1_           ___           ___             ___
 * 3- ( 1.0 2.0)      ___           _1_           ___             ___

Entropy : 1.384703516960144
```

Figure 4.3. The split operation leads the automaton to escape from the local minima.

As seen in Figure 4.3, the entropy of the automaton is increased from 0.69 to 1.38, then decreased to 0.0. In fact, until the split operation the entropy was always 0.69. Namely the automaton was at a local minima. The split operation saved the automaton from this local minima and lead it toward the global minima, whose entropy is 0.0.



```
 *                          0- 3.0              1- 4.0              2- ( 1.0 2.0)       3- ( 1.0 2.0)
 * 0- 3.0                   ___                 ___                 _1_                 _1_
 * 1- 4.0                   ___                 ___                 _1_                 _1_
 * 2- ( 1.0 2.0)            _1_                 ___                 ___                 ___
 * 3- ( 1.0 2.0)            ___                 _1_                 ___                 ___

Entropy : 1.380849540233612
Action  :       GROUPING
----
 *                          0- ( ( ( 1.0 2.0) 3.0) ( ( 1.0 2.0) 4.0))
 * 0- ( ( ( 1.0 2.0) 3.0) ( ( 1.0 2.0) 4.0))___

Entropy : 0.0
```

Figure 4.4. The final automaton, algorithm reached to the global minima.

After escaping from local minima as seen in Figure 4.4 the search resulted to the global minima automaton. In fact this automaton has states those have self recursions, and it is not the exact global minima. However, because the two sequences mixed in random, the count of recursion at each pass is not obvious, so this automaton is the final automaton. Otherwise the learning would continue as overfitting. Exact learning will result failure in such problems, the “OR” problems.

4.2. COMPLEXITY OF THE LEARNING

The complexity of the learning increases when the alphabet size decreases, and the rate of noise increases. Next we will inspect the effect of the alphabet size, and the period size in the success of learning. When the alphabet size gets less, we wait the complexity to grow.

Alphabet Size	Period Size	Noise: 0	5	10	15	20	25	30	35
3	3	99	100	100	98	98	99	99	100
3	6	90	72	53	59	77	82	69	62
3	9	29	29	42	49	33	39	42	31
3	12	12	20	12	8	6	8	9	12
6	3	98	100	100	100	98	99	98	100
6	6	82	99	100	89	98	99	82	92
6	9	79	98	99	91	100	99	89	81
6	12	78	86	73	84	78	100	70	82
9	3	99	100	100	98	100	99	100	100
9	6	100	100	100	99	100	100	100	99
9	9	100	82	100	100	75	100	100	100
9	12	100	89	99	98	92	98	98	95
12	3	99	100	99	100	99	100	100	100
12	6	100	100	100	100	98	100	100	99
12	9	89	100	100	84	81	98	100	99
12	12	98	88	85	93	99	100	99	100

Table 4.1. Results of learning for various alphabet, period sizes and noise amounts.

In this test we used alphabet size of 3 to 12 with step size of 3. For each alphabet size, period size of 3 to 12 with step size of 3 is generated. For each period size we generated periods with noise amounts of 0 to 35 with step size of 5. Each test case is run 100 times and the result of them are given in Table 4.2.

Alp. Size / Per. Size	12	9	6	3
3	13.38	36.75	70.5	98.8
6	81.38	92	92.63	99.13
9	96.13	94.63	99.75	99.5
12	95.25	93.88	99.63	99.65

Table 4.2. Success rate of the algorithm averaged for various noise values.

Table 4.2 shows that, when the period size increases from 3 to 12 with alphabet size of 3, the rate of success shrinks from 100 percent to 6 percent. However, when the alphabet size is 12, period size of 3 and 12 does not show big success rate difference. In Table 4.2 the results for different noise values are averaged. When the alphabet size decreases the algorithm fails to get successful results more frequently. The noise has adverse effect on the learning. However, when the noise threshold set to an appropriate value, the algorithm can cope with the adverse effect of the noise up to some point. As in all signal processing problems, the effect of noise can be eliminated, if the real signal's strength is more than the strength of the noise. However, when the strength of noise gets nearly equal to the strength of the correct period, and if the noise threshold is not tuned well, the noise in the data badly effects the success rate.

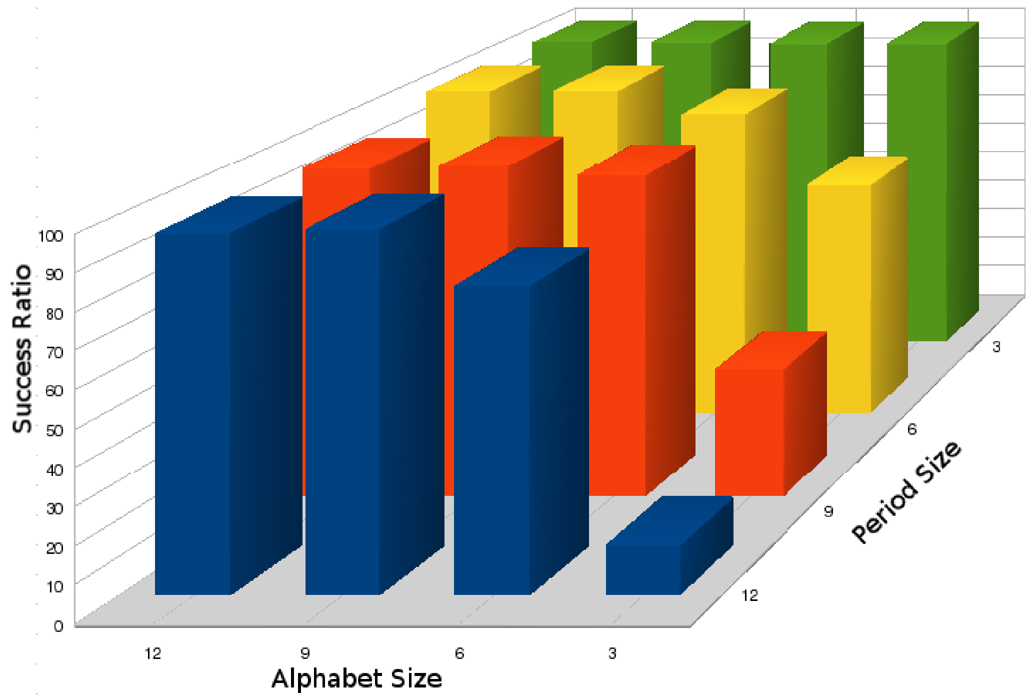


Figure 4.5. The results of the learning algorithm for various alphabet, and period sizes and noise amounts.

As seen in Figure 4.5, when the problem to be solved is a hard problem, increasing the size of alphabet may help the learning algorithm. However, this will effect the generalization of the algorithm adversely. A good balance between them should be achieved, so appropriate values will alleviate the effect of noise, and effect of having a small alphabet.

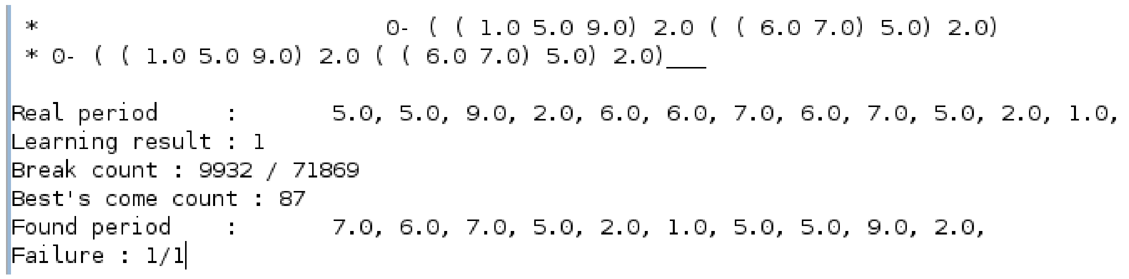

```
 *                          0- ( ( 1.0 5.0 9.0) 2.0 ( ( 6.0 7.0) 5.0) 2.0)
 * 0- ( ( 1.0 5.0 9.0) 2.0 ( ( 6.0 7.0) 5.0) 2.0)___

Real period     :      5.0, 5.0, 9.0, 2.0, 6.0, 6.0, 7.0, 6.0, 7.0, 5.0, 2.0, 1.0,
Learning result : 1
Break count : 9932 / 71869
Best's come count : 87
Found period    :      7.0, 6.0, 7.0, 5.0, 2.0, 1.0, 5.0, 5.0, 9.0, 2.0,
Failure : 1/1
```

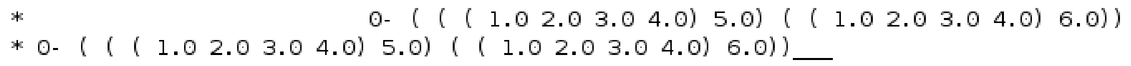
Figure 4.6. Learning with generalization finds a correct automaton, however it could not be tracked successfully.

Figure 4.6 shows an example failure. The found automaton is "((1.0 5.0 9.0) 2.0 ((6.0 7.0) 5.0) 2.0)", and result of the tracker is "(7.0, 6.0, 7.0, 5.0, 2.0, 1.0, 5.0, 5.0, 9.0, 2.0)", but the original period is "(5.0, 5.0, 9.0, 2.0, 6.0, 6.0, 7.0, 6.0, 7.0, 5.0, 2.0, 1.0)" There are different failure types. Some failures are the result of being not able to find any period, like growing too much, having no remaining state, passing the elapsed time threshold for learning. These failures are noncurable failures. However, some failures are because of the result of the generalization. As seen in Figure 4.6, the found automaton also includes the real period in its solution space. Each state and each group, shown in brackets, have self loops. It is easy to track the real period on the found automaton. In the previous work [24], all possible periods, that is in the universe of the found automaton is tested with the input data, and the most successful one is proposed as the found period. However, in our approach we are trying to find the automaton by learning, not by attacking in a brute force way. So at the end of learning, the algorithm again does not make the brute force search either. This rarely results to the cases shown in Figure 4.6. However, from this, we see the successful generalization property of the algorithm.

4.3. TWO SEQUENCES MIXED IN RANDOM - THE "OR" PROBLEM

If two periods are mixed in random, after getting period it will be probable for both of the cases; getting the same period or the other period next. So, there will be an "OR" condition, that will lead to the first period or the second. Because the powers of the self loops are not certain, the learning should be halted in a level. If the learning continues, then the automaton will start to overfit to the data.

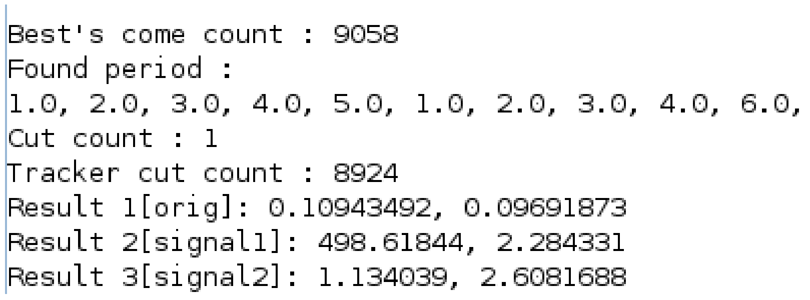
Next, we tested the learning algorithm with a mixture of periods; *Period A is "1, 2, 3, 4, 5" and* Period B is "1, 2, 3, 4, 6". So the period to be learned is random mix of these two period, so it is like "**(A*B*)***".



```
*                    0- ( ( ( 1.0 2.0 3.0 4.0) 5.0) ( ( 1.0 2.0 3.0 4.0) 6.0))
* 0- ( ( ( 1.0 2.0 3.0 4.0) 5.0) ( ( 1.0 2.0 3.0 4.0) 6.0))___
```

Figure 4.7. Resultant automaton for the “OR" problem.

As seen in Figure 4.7, the resultant automaton has structures like “((1234)5)” and “((1234)6)”. Because the learning is done in non-exact mode, each pharanthesis shows a self recursion. So the whole result is like getting “1234” and then “5|6”. The algorithm successfully learned the combination of two periods in random.



```
Best's come count : 9058
Found period :
1.0, 2.0, 3.0, 4.0, 5.0, 1.0, 2.0, 3.0, 4.0, 6.0,
Cut count : 1
Tracker cut count : 8924
Result 1[orig]: 0.10943492, 0.09691873
Result 2[signal1]: 498.61844, 2.284331
Result 3[signal2]: 1.134039, 2.6081688
```

Figure 4.8. Test results of signals and their mixture.

If we further inspect the results of the tests, the original signal’s error rate is less than the two signals’ error rate those composing it. The original signal resulted 0.109, first period resulted 498.618, and the second period resulted to 1.134. So the resultant automaton is capable of classifying the two signals mixed and the mixture of them. This shows the power of the algorithm better. Because we have the second signal as the end of the found period, error rate of second period is less than the error rate of the first period. A better error rate calculation can be devised to calculate the error rates.

4.4. CLASSIFICATION OF SIMILAR PATTERNS

In pattern recognition there is no such case that an algorithm with some specific parameters is the best for each problem. According to the requirements of the problem, each algorithm should be fine-tuned. It is valid for the classification of similar patterns with the resultant automaton of ASPRM.

When the problem is an "OR" problem, it is better to neglect some portion of the differencies. However, when the problem is a classification problem, the aim of the classification should be the key factor for the level of learning. If the automaton will be used for classification of periods those are not very similar, then simplifications like self loops will give better results, because the simple structures will be more robust to the noise. However, if the automaton that is the result of the learning will be used for classification of the similar periods, then such simplifications will lead the result to be not able to classify the periods. So, we should omit these simplifications and search for the exact result when the problem is classifying the highly similar periods. In this case, the of noise will be high, and the ratio of finding the correct automaton will be less. However, when the correct automaton is found, it will successfully classify very similar patterns. This is the problem stated in the previous sections about effects of the ambiguity.

4.4.1. Generation of Similar but Different Periods

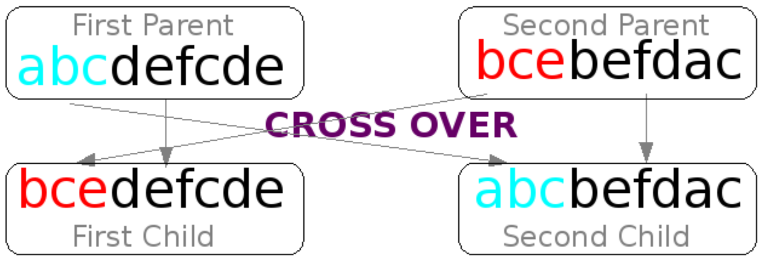


Figure 4.9. Crossover operation leads to similar but different periods.

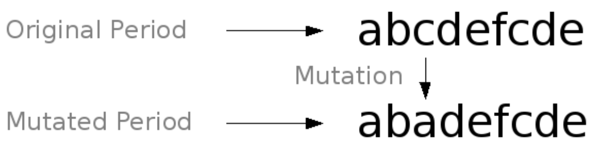


Figure 4.10. Mutation operation leads to similar but different periods.

To generate similar periods, the mutation, in Fig. 4.10, and crossover, in Fig.4.9, operations of genetic algorithms are applied. From the original signal we generated next generations by applying crossover or mutation operations as in Fig.4.11. Each mutated period is different from the original period and also similar to it. As the next generations are generated, the similarity decreases in average and classification of this dissimilar periods from the original signal gets easier.

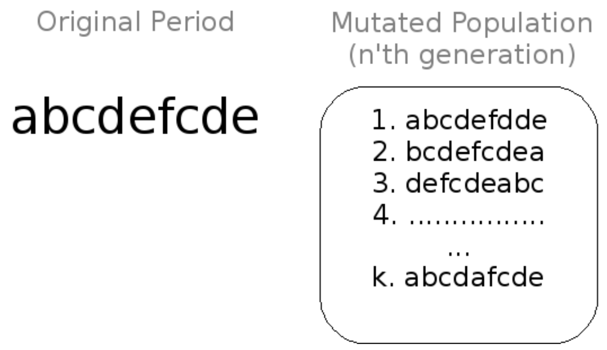


Figure 4.11. Mutated generations, periods those are similar to the original signal but different.

4.4.2. Classification of Similar Periods

A learning process with the original signal is performed and an automaton with the probabilities of the states and the transitions is achieved. Then, each similar but different period, generated by the GA is tested with this automaton as seen in Figure 4.12.

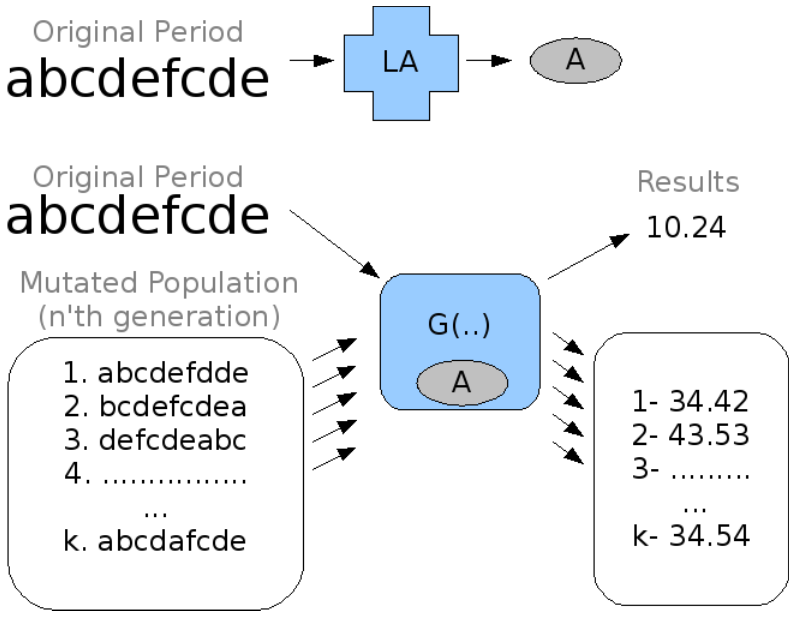
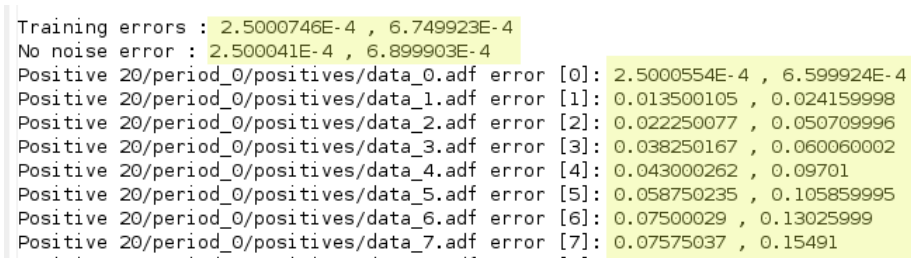


Figure 4.12. Learned automaton is used for classification of the similar but different periods.

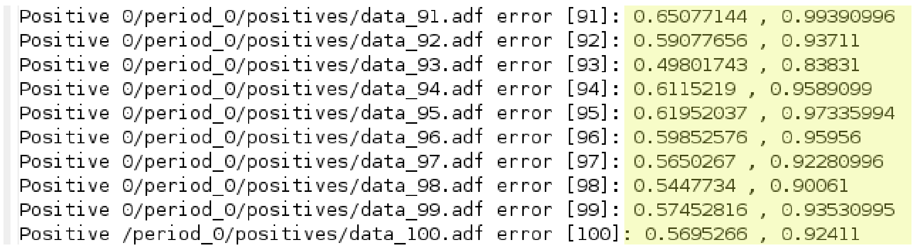
Tree test sets are generated. The first set is the original signal, with noise addition. The first data in this set has no noise, and the last signal has 99% noise, and noise is incremented by one. The second set contains data of mutated periods with no noise. The first ten periods are from the first generation and the last ten periods are from the tenth generation, and for each ten the generation is increased in between. The last set of inputs are the same with the second set, but they are noise added version of the second data set.



```
Training errors : 2.5000746E-4 , 6.749923E-4
No noise error : 2.500041E-4 , 6.899903E-4
Positive 20/period_0/positives/data_0.adf error [0]: 2.5000554E-4 , 6.599924E-4
Positive 20/period_0/positives/data_1.adf error [1]: 0.013500105 , 0.024159998
Positive 20/period_0/positives/data_2.adf error [2]: 0.022250077 , 0.050709996
Positive 20/period_0/positives/data_3.adf error [3]: 0.038250167 , 0.060060002
Positive 20/period_0/positives/data_4.adf error [4]: 0.043000262 , 0.09701
Positive 20/period_0/positives/data_5.adf error [5]: 0.058750235 , 0.105859995
Positive 20/period_0/positives/data_6.adf error [6]: 0.07500029 , 0.13025999
Positive 20/period_0/positives/data_7.adf error [7]: 0.07575037 , 0.15491
```

Figure 4.13. Result values for original signal with low noise values.

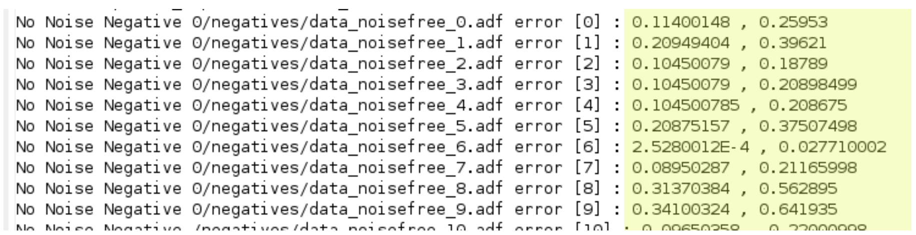
Figure 4.13 shows the results of testing the learned automaton with data of the mutated periods in the first generation. Because the noise rate is low, the error rates are quite low too.



```
Positive 0/period_0/positives/data_91.adf error [91]: 0.65077144 , 0.99390996
Positive 0/period_0/positives/data_92.adf error [92]: 0.59077656 , 0.93711
Positive 0/period_0/positives/data_93.adf error [93]: 0.49801743 , 0.83831
Positive 0/period_0/positives/data_94.adf error [94]: 0.6115219 , 0.9589099
Positive 0/period_0/positives/data_95.adf error [95]: 0.61952037 , 0.97335994
Positive 0/period_0/positives/data_96.adf error [96]: 0.59852576 , 0.95956
Positive 0/period_0/positives/data_97.adf error [97]: 0.5650267 , 0.92280996
Positive 0/period_0/positives/data_98.adf error [98]: 0.5447734 , 0.90061
Positive 0/period_0/positives/data_99.adf error [99]: 0.57452816 , 0.93530995
Positive /period_0/positives/data_100.adf error [100]: 0.5695266 , 0.92411
```

Figure 4.14. Result values for original signal with high noise values.

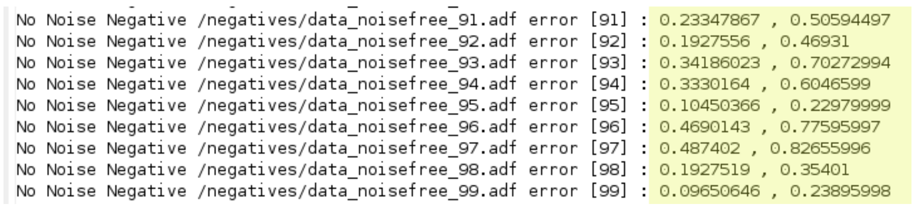
As seen in Figure 4.13 and Figure 4.14, when the amount of noise increases, the test algorithm, with the learned automaton and probabilities, produces higher error rate values.



```
No Noise Negative 0/negatives/data_noisefree_0.adf error [0] : 0.11400148 , 0.25953
No Noise Negative 0/negatives/data_noisefree_1.adf error [1] : 0.20949404 , 0.39621
No Noise Negative 0/negatives/data_noisefree_2.adf error [2] : 0.10450079 , 0.18789
No Noise Negative 0/negatives/data_noisefree_3.adf error [3] : 0.10450079 , 0.20898499
No Noise Negative 0/negatives/data_noisefree_4.adf error [4] : 0.104500785 , 0.208675
No Noise Negative 0/negatives/data_noisefree_5.adf error [5] : 0.20875157 , 0.37507498
No Noise Negative 0/negatives/data_noisefree_6.adf error [6] : 2.5280012E-4 , 0.027710002
No Noise Negative 0/negatives/data_noisefree_7.adf error [7] : 0.08950287 , 0.21165998
No Noise Negative 0/negatives/data_noisefree_8.adf error [8] : 0.31370384 , 0.562895
No Noise Negative 0/negatives/data_noisefree_9.adf error [9] : 0.34100324 , 0.641935
No Noise Negative /negatives/data_noisefree_10.adf error [10] : 0.09650358 , 0.22999998
```

Figure 4.15. Result values for first mutated generation.

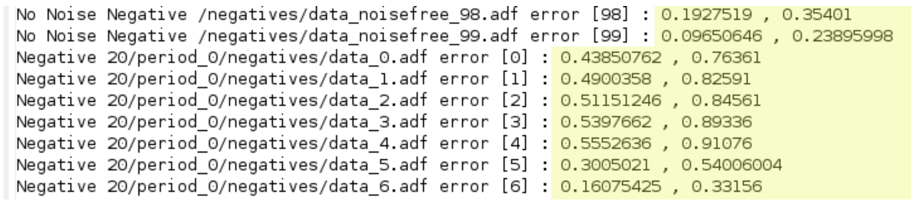
Figure 4.15 shows the results of testing the learned automaton with data of the mutated periods in the first generation. The error rates are not much for them.



```
No Noise Negative /negatives/data_noisefree_91.adf error [91] : 0.23347867 , 0.50594497
No Noise Negative /negatives/data_noisefree_92.adf error [92] : 0.1927556 , 0.46931
No Noise Negative /negatives/data_noisefree_93.adf error [93] : 0.34186023 , 0.70272994
No Noise Negative /negatives/data_noisefree_94.adf error [94] : 0.3330164 , 0.6046599
No Noise Negative /negatives/data_noisefree_95.adf error [95] : 0.10450366 , 0.22979999
No Noise Negative /negatives/data_noisefree_96.adf error [96] : 0.4690143 , 0.77595997
No Noise Negative /negatives/data_noisefree_97.adf error [97] : 0.487402 , 0.82655996
No Noise Negative /negatives/data_noisefree_98.adf error [98] : 0.1927519 , 0.35401
No Noise Negative /negatives/data_noisefree_99.adf error [99] : 0.09650646 , 0.23895998
```

Figure 4.16. Result values for tenth mutated generation.

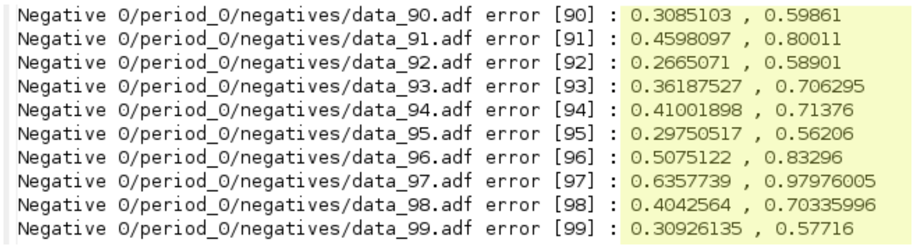
Figure 4.16 shows the results of testing the learned automaton with data of the mutated periods in the tenth generation. The error rates are much higher than the error rates for the first generation.



```
No Noise Negative /negatives/data_noisefree_98.adf error [98] : 0.1927519 , 0.35401
No Noise Negative /negatives/data_noisefree_99.adf error [99] : 0.09650646 , 0.23895998
Negative 20/period_0/negatives/data_0.adf error [0] : 0.43850762 , 0.76361
Negative 20/period_0/negatives/data_1.adf error [1] : 0.4900358 , 0.82591
Negative 20/period_0/negatives/data_2.adf error [2] : 0.51151246 , 0.84561
Negative 20/period_0/negatives/data_3.adf error [3] : 0.5397662 , 0.89336
Negative 20/period_0/negatives/data_4.adf error [4] : 0.5552636 , 0.91076
Negative 20/period_0/negatives/data_5.adf error [5] : 0.3005021 , 0.54006004
Negative 20/period_0/negatives/data_6.adf error [6] : 0.16075425 , 0.33156
```

Figure 4.17. Result values for first mutated generation with noise addition.

Figure 4.17 shows the results of testing the learned automaton with noise added data of the mutated periods in the first generation. The error rates are higher than the error rates for the data of the first generation with no noise.

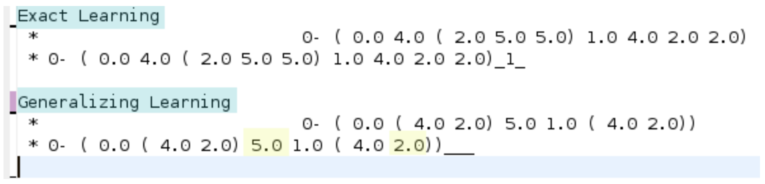


```
Negative 0/period_0/negatives/data_90.adf error [90] : 0.3085103 , 0.59861
Negative 0/period_0/negatives/data_91.adf error [91] : 0.4598097 , 0.80011
Negative 0/period_0/negatives/data_92.adf error [92] : 0.2665071 , 0.58901
Negative 0/period_0/negatives/data_93.adf error [93] : 0.36187527 , 0.706295
Negative 0/period_0/negatives/data_94.adf error [94] : 0.41001898 , 0.71376
Negative 0/period_0/negatives/data_95.adf error [95] : 0.29750517 , 0.56206
Negative 0/period_0/negatives/data_96.adf error [96] : 0.5075122 , 0.83296
Negative 0/period_0/negatives/data_97.adf error [97] : 0.6357739 , 0.97976005
Negative 0/period_0/negatives/data_98.adf error [98] : 0.4042564 , 0.70335996
Negative 0/period_0/negatives/data_99.adf error [99] : 0.30926135 , 0.57716
```

Figure 4.18. Result values for tenth mutated generation with noise addition.

Figure 4.18 shows the results of the testing the learned automaton with noise added data of the mutated periods in the tenth generation. The error rates are quite higher than the error rates for the data of the first generation with no noise.

As seen in Figure 4.15 and Figure 4.16, if we go further to the next generations, the test algorithm, with the learned automaton and probabilities, produces higher error rate values. If noise is added to the mutated periods, the error rates increase to the higher values, as seen in Figure 4.17 and Figure 4.18.



```
Exact Learning
 *                        0- ( 0.0 4.0 ( 2.0 5.0 5.0) 1.0 4.0 2.0 2.0)
 * 0- ( 0.0 4.0 ( 2.0 5.0 5.0) 1.0 4.0 2.0 2.0)_1_

Generalizing Learning
 *                        0- ( 0.0 ( 4.0 2.0) 5.0 1.0 ( 4.0 2.0))
 * 0- ( 0.0 ( 4.0 2.0) 5.0 1.0 ( 4.0 2.0))___
```

Figure 4.19. Resultant automatons of exact learning and learning with generalization.

As seen in Figure 4.19, when exact learning is applied, the full sequence is learned as the target automaton. However, it may not be required if the classification need not to be very accurate. If learning is done with generalization, then it will be more robust to noise in the learning phase. So, user must make a decision about whether learning to be more robust to noise, or the learned automaton to be a better classifier.

5. CONCLUSIONS

In this work we focus on generation of Variable order Markov Models (VMMs) over a finite alphabet $\sum$. In contrast to other fixed context length models, like N-gram Markov models, our algorithm is capable of learning conditional distributions of the form $P(\sigma|s)$, with $s \in \sum^N$, and $\sigma \in \sum$ where context length $\mid s \mid$ varies. With this property the algorithm is capable of capturing both large and small order Markov dependencies according to the input data. Adaptively learning the context length saves the algorithm from specifying one of the most important super variable, the order of learning.

We tested the algorithm for various alphabets, noise ratios, and period lengths. Furthermore we generated quite similar periods and tested the learning capability of algorithm. As we noted, the complexity of the learning increases when the alphabet size decreases, and the rate of noise increases. However, some improvements can be made to the algorithm. A better preprocessing module can be added to the design. Algorithms like fuzzy string matching may lead the algorithm to be more robust to the noise. We tested the algorithm for ECG data, and will continue these tests. The algorithm may be tested for the signal sequences from different areas like NLP.

REFERENCES

[1] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *"Introduction To Automata Theory, Languages, And Computation"*. The Addison-Wesley press, **2001**.

[2] K. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction.* Prentice Hall, Inc., **1989**.

[3] R. Sutton and A. Barto, *Reinforcement Learning: an Introduction.* MIT Press, **1999**.

[4] K.-Y. Huang, "Syntactic pattern recognition for seismic oil exploration," *Series in Machine Perception and Artificial Intelligence*, vol. 46, pp. 1–5, **2002**.

[5] A. USTIMOV, "Cycle detection in noisy signals by constructive automata: An adaptive syntactic approach to pattern recognition," Master's thesis, Marmara University, Istanbul, Turkey, **2005**.

[6] E. Alpaydin, *"Introduction to Machine Learning"*. Cambridge, Massachusetts, London, England: The MIT Press, **2006**.

[7] N. Baba and Y. Mogami, "A new learning algorithm for the hierarchial structure learning automata operating in the nonstationary s-model random environment," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 32, no. 6, pp. 750–757, **2002**.

[8] B. Demiröz and M. B. Tümer, "Signal compression using growing cell structures: A transformational approach," in *Computer and Information Sciences - ISCIS 2003, 18th International Symposium*, (Antalya, Turkey), **2003**.

[9] S. Alp and H. L. Akın, "Kalman based finite state controller for partially observable domains," *International Journal of Advanced Robotic Systems*, vol. 3, no. 4, pp. 331–342, **2006**.

[10] R. S. Sutton and A. G. Barto, *"Reinforcement Learning: An Introduction"*. Cambridge, Massachusetts, London, England: A Bradford Book, The MIT Press, **1998**.

[11] D. Blostein and A. Grbavec, "Recognition of mathematical notation," in *Handbook of Character Recognition and Document Image Analysis* (E. H. Bunke and P. Wang, eds.), p. Handbook of Character Recognition and Document Image Analysis, **1997**.

[12] B. Moayer and K. S. Fu, "A syntactic approach to fingerprint pattern recognition," *Pattern Recognition*, vol. 7, pp. 1–23, **1975**.

[13] A. Pérez, M. I. Torres, and F. Casacuberta, "Speech translation with phrase based stochastic finite-state transducers," in *In Proceedings of the 32nd International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2007)*, vol. IV, (Honolulu, Hawaii USA), pp. 113–116, **Apr 2007**.

[14] A. Koski, M. Juhola, and M. Meriste", "Syntactic recognition of ECG signals by attributed finite automata," *"Pattern Recognition"*, vol. 28, pp. 1927–1940, **1995**.

[15] P. Trahanias, E. Skordalakis, and G. Papakonstantinou, "A syntactic method for the classification of the QRS patterns," *Pattern Recognition Letters*, vol. 9, pp. 13–18, **1989**.

[16] P. Trahanias and E. Skordalakis, "Syntactic pattern recognition of the ECG," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 648–657, **1990**.

[17] V. Reddy and R. Narasimhan, “Some experiments in scene analysis and scene regeneration using compax,” *CGIP*, vol. 1, pp. 386–393, **December 1972**.

[18] V. Shankar, J. Rodriguez, and M. Gettings, “Texture analysis for automated classification of geologic structures,” in *Southwest06*, pp. 81–85, **2006**.

[19] H. Liu and K. Fu, “A syntactic approach to seismic pattern recognition,” *PAMI*, vol. 4, pp. 136–140, **March 1982**.

[20] H. Liu and K. Fu, “A syntactic pattern recognition approach to seismic discrimination,” *Geoexploration*, vol. 20, pp. 183–196, **1982**.

[21] T. O’Haver, “Introduction to signal processing - smoothing.” http://www.wam.umd.edu/ toh/spectrum/Smoothing.html, Oct 2007.

[22] S. Prog, “Low frequency butterworth and optimal wiener ecg filters.” http://www.scienceprog.com/low-frequency-butterworth-and-optimal-wiener-ecg-filters/, Oct 2007.

[23] “Electrocardiogram.” http://en.wikipedia.org/wiki/Electrocardiogram, Oct 2007.

[24] A. Ustimov and M. B. Tümer, “Construction of a learning automaton for cycle detection in noisy data sequences,” in *The 20th International Symposium on Computer and Information Sciences*, (Istanbul, Turkey), **2005**.

CURRICULUM VITAE

He was born in Ankara in 1982. He completed his primary education in Osman Unyazici Primary School, and secondary education in Altinordu Secondary School. After secondary education, he finished Sincan High School. In 1999 he was admitted to Marmara University, Faculty of Engineering, Department of Computer Engineering. He graduated in 2004 with a degree of BSc. After, he started to MSc degree in the same field in Marmara University.