DESIGN AND IMPROVEMENT OF MULTI-LEVEL DECISION-MAKING MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ULAŞ BELDEK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JUNE 2009

Approval of the thesis:

DESIGN AND IMPROVEMENT OF MULTI-LEVEL DECISION-MAKING MODELS

submitted by **ULAŞ BELDEK** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Kemal Leblebicioğlu _____
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members

Prof. Dr. İsmet Erkmen _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Kemal Leblebicioğlu _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Faruk Polat _____
Computer Engineering Dept., METU

Prof. Dr. Uğur Halıcı _____
Electrical and Electronics Engineering Dept., METU

Asst. Prof. Dr. Reza Hassanpour _____
Computer Engineering Dept., Çankaya University

Date: 17.06.2009

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: **Ulaş Beldek**

Signature:

ABSTRACT

DESIGN AND IMPROVEMENT OF MULTI-LEVEL DECISION-MAKING MODELS

Beldek, Ulaş
Ph.D., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Kemal Leblebicioğlu

June 2009, 218 Pages

In multi-level decision making (DM) approaches, the final decision is reached by going through a finite number of DM levels. Usually, in each level, a raw decision is produced first and then a suitable decision fusion technique is employed to merge the lower level decisions with the raw decision in the construction of the final decision of the present level. The basic difficulty in these approaches is the determination of how the consecutive levels should interact with each other. In this thesis, two different multi-level DM models have been proposed. The main idea in the first model, "hierarchical DM" (HDM), is to transfer the decisions of previous hierarchical levels to an upper hierarchy with some reliability values. These decisions are then fused using a suitable decision fusion technique to attain more consistent decisions at an upper level. The second model "local DM in multiple-levels" (LDM-ML) depends on what may be called as local DM process. Instead of designing an agent to perform globally, designing relatively simple agents which are supposed to work in local regions is the essence of the second idea. Final decision is partially constructed by contribution of a sufficient number of local DM agents. A successful local agent is retained in the agent pool whereas a local agent not successful enough is eliminated and removed from the agent pool. These models have been applied on two case studies associated with fault detection in a four-tank system and prediction of lotto sales.

Keywords: Decision Making, Decision Fusion, Agents, Expert Systems, Optimization.

ÖZ

ÇOK SEVİYELİ KARAR VERME MODELLERİNİN TASARLANMASI VE GELİŞTİRİLMESİ

Beldek, Ulaş
Doktora, Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Kemal Leblebicioğlu

Haziran 2009, 218 sayfa

Çok seviyeli karar verme yaklaşımlarında, son karara sonlu sayıda karar verme seviyelerinden geçilerek ulaşılır. Genellikle, her seviyede, önce bir başlangıç kararı üretilir. Daha sonra uygun bir karar birleştirme tekniği, bulunulan seviyedeki son kararı oluşturmada, alt seviye kararı başlangıç kararı ile birleştirmek için kullanılır. Çok seviyeli yaklaşımlarda temel zorluk ardışık seviyelerin birbirleriyle nasıl etkileşmesi gerektiğinin belirlenmesidir. Bu tezde, iki farklı çok seviyeli karar verme modeli önerilmiştir. İlk modelde (hiyerarşik karar verme) esas fikir, önceki hiyerarşik seviye kararlarının bir üst seviyeye bazı güvenilirlik değerleri ile aktarılmasıdır. Daha sonra bir üst seviyedeki karar, bu kararların uygun bir karar birleştirme tekniği ile birleştirilmesi ile gerçekleştirilir. İkinci model (Çoklu seviyelerde yerel karar verme) yerel karar verme diye adlandırılan bir kavrama dayanır. Evrensel çalışan bir ajan tasarlamak yerine, göreceli olarak yerel bölgelerde çalışan nispeten daha basit ajanlar tasarlamak ikinci fikrin esasıdır. Son karar, kısmen, yeterli sayıda yerel karar verme ajanın katkısıyla inşa edilir. Başarılı bir yerel ajan kullanılmaya devam edilirken yeterince başarılı olamayan bir ajan elenir ve ajan havuzundan çıkarılır. Geliştirilen modeller dört-tanklı sistemde hata bulma ve lotto satışı tahmini problemlerine uygulanmıştır.

Anahtar Kelimeler: Karar Verme, Karar Birleştirme, Ajanlar, Uzman Sistemler, Eniyileme.

To My Parents;

Hülya and Mustafa BELDEK

Aileme;

Hülya ve Mustafa BELDEK

ACKNOWLEDGEMENTS

I want to declare the names of a few people who assisted me throughout the preparation of this thesis. First of all I wish to express my deepest gratitude to my supervisor Prof. Dr. Kemal Leblebicioğlu for his valuable guidance and comments. I want to thank to my parents Hülya Beldek and Mustafa Beldek. They were always with me in my troubled times. I also want to express my great appreciation for Prof. Dr. Kenan Taş, Prof. Dr. Ümit Yüceer, Asst. Prof. Dr. Orhan Gazi, Öncü Hazır, Inst. Larry D. White, Selin Akay, Tolga İnan, Utkan Eryılmaz, Orkunt Sabuncu, Evren Deviren, for their very valuable support, especially in proof reading of several chapters of the thesis. I also want to thank Çankaya University for backing me up. Besides, I also want to state my regards to many people whom I did not mention here.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CS1 : Case Study 1

CS2 : Case Study 2

DM : Decision-Making

FL : Fuzzy Logic

GA : Genetic Algorithm

GP : Genetic Programming

HDM : Hierarchical Decision-Making

LDM-ML : Local Decision-Making in Multiple-Levels

NB : Negative Big

NM : Negative Medium

NN : Neural Network

NS : Negative Small

PB : Positive Big

PM : Positive Medium

PS : Positive Small

SNR : Signal-to-noise ratio

Z : Zero

CHAPTER 1

INTRODUCTION

1.1. What is Decision-Making?

Decision-making (DM) is the cognitive process of selecting an outcome or an action from the list of several possible alternatives. The outcome of the DM process is a decision in response to the stimuli. For this reason, it can be explained as a reasoning process to deal with current problems based on the overall sensory data and the present memory of the system [1].

1.2. What is a Decision-Making System?

DM systems are organized procedural structures with memory that collect and evaluate available information for the DM problem engaged to infer decisions as a reaction to, criticism of or opinion on the collected sensory data. Depending on the complexity of the DM process, the architecture of a DM system may consist of single or multiple experts which themselves are DM systems of lower complexity. The design of DM systems to produce consistent decisions for complicated problems is very hard or nearly impossible. It is a common practice to separate the process into circumstances for which it is relatively simple to construct a DM unit called an expert. Decision on the design of the overall system then is based on the interaction of the different experts as well as the information obtained. In literature, there are several approaches on the partitioning and fusion of the expert decisions in order to construct DM systems [2].

1.3. Decision-Making Systems

Most of the standard DM systems depend on assessing the relative importance of alternatives. Analytic hierarchy process [3] is such an example. Possible decision choices are compared using a list of criteria and are ordered hierarchically. The possible choices are evaluated and their suitability to satisfy each criterion is assessed. A weighted averaging operation determines the influence of each criterion.

Different from the classical DM systems and benefiting from specialized procedures, expert systems make use of several decision makers. Each expert has a different degree of influence to determine the best decisions. In expert system design, the first stage is the determination of the structure and the number of experts. The second stage is the way in which the experts interact with each other to work towards the construction of the decision. The main issue when multiple experts exist is how to specify the relationship between the experts [4] (i.e., determining the style of how the group DM is performed). Many different group DM systems exist depending on how the experts interact with each other: Providing a consensus among experts having varying amounts of dominance in group DM is studied in [5]. Several systems depend on voting or dominance of a group of experts (i.e., majority opinion) [6] instead of consensus DM: weighted averaging methods to determine the majority opinion [4] are the most widely-used. Such approaches use two methods to create the majority opinion. Expert opinions are combined via some aggregation operators to obtain the majority opinion or the majority opinion is assessed as an imprecise value having a fuzzy nature so that it is represented as a fuzzy value with an indication of its strength. [7] deals with two methods to predict and adjust the weights of fuzzy opinions. The methods either employ the minimization of the sum of the squared distances between the weighted fuzzy opinions or the minimization of the difference between defuzzied values of the weighted fuzzy opinions. Multi-criteria DM techniques try to create a regulation such that each criterion is somewhat relaxed from the desired levels. In [8] a multi-criteria, multi-expert group DM example is studied. The preferences of

the experts for the alternatives are demonstrated by a list of values. These lists of values are multiplied by the weights to assess the relative importance of each different criterion.

Preference aggregation of experts in group DM applications can be managed in many ways. Different data fusion and evaluation methods can be used for this purpose. In [9] a framework for DM is set whose fundamentals depend on optimal aggregation of fuzzy concepts.

Resolving uncertainties is another important issue in DM [10-13]. One of the uncertainties is the case when experts possess insufficient amount of knowledge about the problem requiring DM [10]. To demonstrate the preferences of the experts, intuitionistic (depending on intuition) preference relations [10] are put forward by using intuitionistic fuzzy sets and afterwards methods employing these relations are applied to group DM. [11] deals with the update of preferences of decision makers when attribute weights are available to a degree. The main concern in [12] is the evaluation and manipulation techniques of data from various knowledge sources which possess different domains and scales such that reducing them to a single domain has the potential to augment uncertainty. Fuzzy multi-attribute DM in uncertainty [13] is another problem. The proposed model in [13] reflects both qualitative and quantitative opinions of decision makers.

In group DM applications mentioned until now, the relations between experts are horizontal (associated in a single level). However, it is an important question of what to do if the encountered problem is more complex. Multi-level DM systems are more suitable for complex problems. In multi-level DM applications, at each level of the hierarchy there exist a number of experts that influence the functionality of the other experts at the lower or upper levels. For instance, the interaction between decision makers are organized in [14] in such a way that the followers both influence the decisions of the leaders and also each other. Hierarchical DM systems may permit policy changes in making decisions. [15] puts forward a fuzzy programming method to be used in a hierarchical DM

structure. Decision makers located at different levels have different levels of aspiration to fulfill their objectives. The final decision is achieved using a measure of minimizing the groups' (decision makers) feelings of regret or dissatisfaction.

When a multi-level DM structure is established, the most important issue is the fusion (combination) of the information gained at different levels. In [16], a framework for decision fusion is provided for a hierarchical DM system. The confidence values of the decisions of experts are taken into account in the decision fusion process using some performance indices of the experts.

In some of the group DM systems (either at multiple levels or at a single level), locality information is provided to determine the degree of interaction between the group members or to determine for which regions the group members are responsible [17]. In those applications, local decision makers perform the decision in or around some region of their concern and the decisions of local agents are fused using a suitable fusion (i.e., aggregation) method depending on locality information. To develop local agents, evolutionary algorithms may be employed [18].

Machine learning tools and data evaluation and interpolation techniques are widely used to help design DM systems [19-21]. In several applications experts are structured as trained agents. Rule-bases employing fuzzy logic (FL) for data interpolation, types of neural network (NN) structures trained via various machine learning techniques, support vector machines [20] whose parameters are optimized using genetic algorithm (GA) are typical examples of these agent structures. For example, in [20] a DM problem structured as a classification problem in bioinformatics is handled where a GA is applied to optimize fuzzy feature transformations. Ways of extracting fuzzy decision rules from a fuzzy information system [22] and construction of a genetic fuzzy agent for a scheduling system [23] are other examples of these machine learning applications.

1.4. Problems Encountered in Construction of Decision-Making Systems

Depending on the composition of the DM problem, many different DM systems (i.e., single expert, multiple expert, group DM, multi-level DM, local DM) can be constructed and many different data evaluation techniques and data structures (i.e., NNs, rule-bases, support vector machines) can be used. However, every DM system and every data evaluation technique and data structure has some advantages and disadvantages. DM systems using a data evaluation technique benefit from the advantages of that evaluation technique. Unfortunately, they also possess the disadvantages sides of the technique as well. For example, a rule-base acting [22] as a decision maker has the potential to perform fine decisions if its structure is adequate and compact enough and if it is trained as much as necessary for the DM problem discussed. For instance, to develop a rule-base having a huge number of rules in its body may be time-consuming. Even in some cases, development of such a rule-base may be a burden for the training procedure. On the contrary, a rule-base having a small number of rules in its body may be effective on a few occasions whereas it may be unsuccessful for the remaining occasions. Similarly, a NN acting as a decision maker has the potential to perform good decisions for training situations due to its ability to evaluate non-linear data. However if it is over-trained, its performance for the validation situations become limited.

We can also check the effectiveness of a DM system by considering its structure (i.e., depending on the style of how DM is has been performed). In group DM, the authority to perform decision is distributed between different group members [4]. This authority distribution provides flexibility. If the interaction between the members (i.e., decision fusion, authority distribution) is well-defined, satisfactory results are achieved. However as the number of authorities increase, building up the relationship between them may become harder. Depending on the complexity of the DM problem, different strategies may be employed. Hierarchical DM systems [14] have the potential to yield better solutions for complex problems having nonlinearities or dependencies. On the contrary, although the degree of

flexibility of hierarchical DM systems is superior compared to the degree of flexibility of single level systems, the success of single level DM systems is more likely if the interactions between hierarchical levels are ill-defined. Hence, a well-organized structure is required to develop hierarchical DM systems.

Another alternative that can be used for complex problems is to use decentralized systems [24] similar to local decision makers. The main difficulty in the construction of local DM systems arises from the establishment of the interaction between local experts, especially when expressing the locality information between the experts. Locality information should be prepared carefully to prevent failure in problem solving.

1.5. Required Properties of Decision-Making Models to be Constructed

Our aim in this thesis is to put forward new DM model(s). This is a design process and as usual, before every design process starts, one has to establish design specifications. Primarily, the most noticeable property desired is flexibility. If some course of action changes is required in the composition of the model, it should be implemented with ease. Additionally, the model should be open to further development. If the model structure is not successful enough according to a user-determined measure of performance, it should be possible to augment or change the structure of the DM model easily to form a new model. The DM model can be applied to complex DM problems where different policies have to be considered together to solve the problem effectively. The properties required for building such a DM model can be summarized as follows:

- The model should have the potential to be used in complex DM problems.
- It should have re-training and development capacity.
- It should have an adaptable and flexible structure which can be projected easily according to the new requirements.

- It should provide an elastic structure enabling the use of different data evaluation techniques, different decision-fusion methods and applicability of various machine learning methods.

Due to these requirements, it is planned to design multi-level DM models. Multi-level DM models do not reach a final decision in a single attempt. Depending on the nature of the problem, they provide relatively primitive decisions at lower levels. As the levels increase, the primitive decisions are combined and modified. These decisions are evolved gradually with increasingly sophisticated and complicated structures as the levels increase. What is planned to be done in this study is to construct and to develop multi-level DM models functioning with this systematic mentioned above. The efficiency of the DM models is investigated with two case study problems.

A multi-level DM model provides flexibility in three ways. First of all, utilization of different fusion techniques at different levels can be possible in such a structure. Different decision fusion techniques can be applied from level to level depending on the policy changes. Secondly, depending on the requirements in a level, a different number of experts may take part in each level's decisions. Lastly, the levels may provide a framework to decompose the problem into more easily-handled sections. Besides, levels also provide an opportunity to apply various data evaluation techniques along with different optimization algorithms.

Unfortunately, there are also some drawbacks when using multi-level DM models. One of them is that the relationship between the experts of different levels should be well-organized (i.e., the amount and the functionality of the experts). This organization is so problem-specific that an organization providing good results in one problem may be inappropriate for another problem setting. If the interactions are not well-organized, it will not be beneficial to use multi-level models. For this reason, the flexibility of the multi-level DM models can account for their disadvantageous aspects as well.

1.6. The Case Study Problems

A dynamic system (a water circulation system with four water tanks [25]) that has nonlinearities in its structure is used as the first case study problem (CS1). Some failures are created artificially in the system structure by changing several parameters. Two sets of measurements are taken for the variables of the system: One uses the original system dynamics and the other one uses the system dynamics with parameter changes. An error data is constructed as the difference between the two sets of measurements. The error data is used as the input to develop and examine the multi-level DM models whose job are to predict the related parameter changes in the system structure in different conditions of failure. From a different perspective, CS1 is a fault detection problem where the aim is to determine the fault characteristics (the change in the parameters) using the proposed multi-level DM models. Specifically, the multi-level DM models classify the characteristics of the different faults according to the error data.

There are two motivations to use CS1 as a test bed for the DM models that are proposed in this thesis. Firstly, the system has a complex and a nonlinear structure; variations created in a single parameter may affect more than one variable of the error data. This is due to the coupled structure of the system variables. A fluctuation in a variable caused by a parametric change may result in fluctuations in other variables. Secondly, the structure of the system is suitable for the creation of multiple faults: A number of parameter changes may be performed at different instants which make the fault detection process more complicated. Furthermore, multiple fault situations combined with the coupled structure of the system makes the fault detection process more demanding. When all of these are considered, CS1 is almost a perfect test bed to develop complex DM models.

The second case study problem (CS2) is a Lotto analysis problem. The aim in CS2 is to predict the number of sales of tickets at present drawing using the past and present values of lotto drawings (the announced prize and the price of a lotto

ticket). The details of CS2 and application of multi-level DM models to this problem are presented in Chapter 8.

1.7. Proposed Multi-Level Decision-Making Models

There are mainly two different multi-level DM models proposed in this thesis. The first model is called as hierarchical decision making (HDM) model. In this model there may be a single agent in each level or multiple numbers of agents in different hierarchical levels. HDM relies on the transfer of the decisions of the developed agent(s) to subsequent levels according to a performance value. In the first level, an agent, which is made up of a rule-base or any structure performing DM, is developed using an optimization technique to determine the first level decisions. The decisions of the developed agent are conveyed to the second level at a specified performance value. The first level decisions with the associated performance values are used in the development of the second level agent(s). The second level agent has two parts: The first part is the dynamic part of the agent which is typically a new DM structure or a new rule-base; the second part is the static part composed of the decisions of the first level agent with their associated performance values. The decisions of the new rule-base to be developed and the first level decisions are fused using an aggregation (fusion) method based on the performance values of first level decisions. Optimization of the structure (rule-base) using the fused decisions constitutes the dynamic part of the second level agent. The resultant decisions at the end of the optimization process are the second level decisions. The relationship between a previous level and the current level is similar to the relationship between the second level and the first level. The hierarchical improvement process continues until a desired degree of success is attained for the decisions. A simple illustration of HDM model is shown in Fig 1.1. The development process of a first level and a second level agent using the HDM model are described in Fig. 1.2 and Fig. 1.3, respectively.

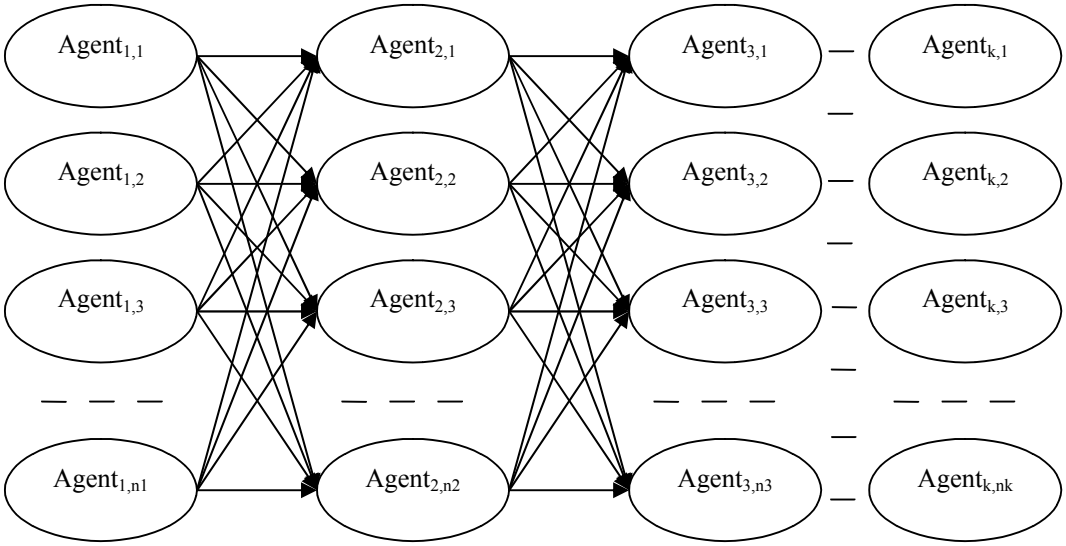

Fig. 1.1: An illustrative scheme for the HDM model. The first level agents help development of the second level agents. The second level agents help development of the third level agents.

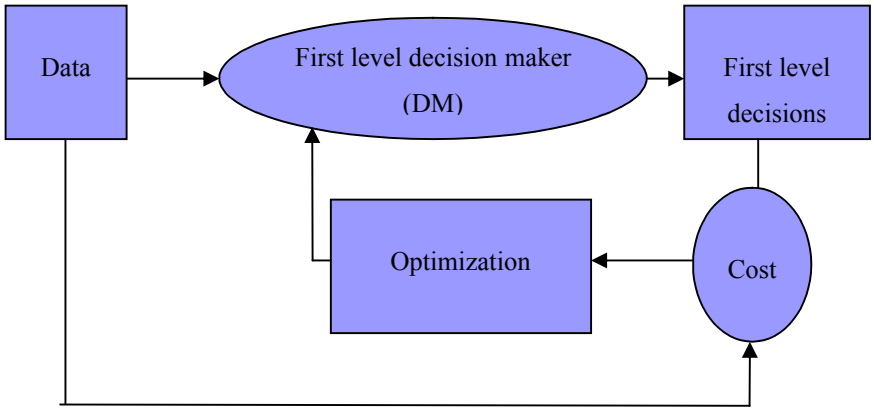

Fig. 1.2: The scheme for the development of the first level decision maker.

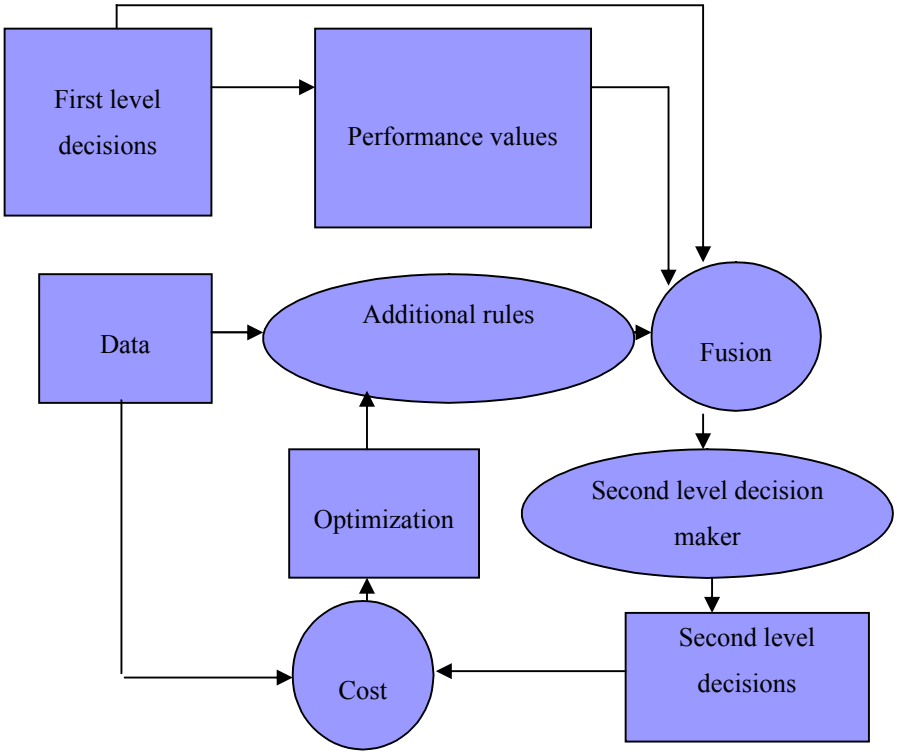

Fig. 1.3: The scheme for the development of the second level decision maker.

It is also possible to generalize HDM model agent development process schemes for multi-agent structures as in Fig. 1.3 (the development of a second level agent using at least two first level agents). More sophisticated hierarchical structures can also be produced regarding the requirements to solve the problem. Different cost functions can be selected in each step so that the hierarchical structure is used for multi-criteria DM applications or different kinds of decision fusion techniques can be applied so that each agent is trained in a different manner and become applicable to different regions of interest at the problem.

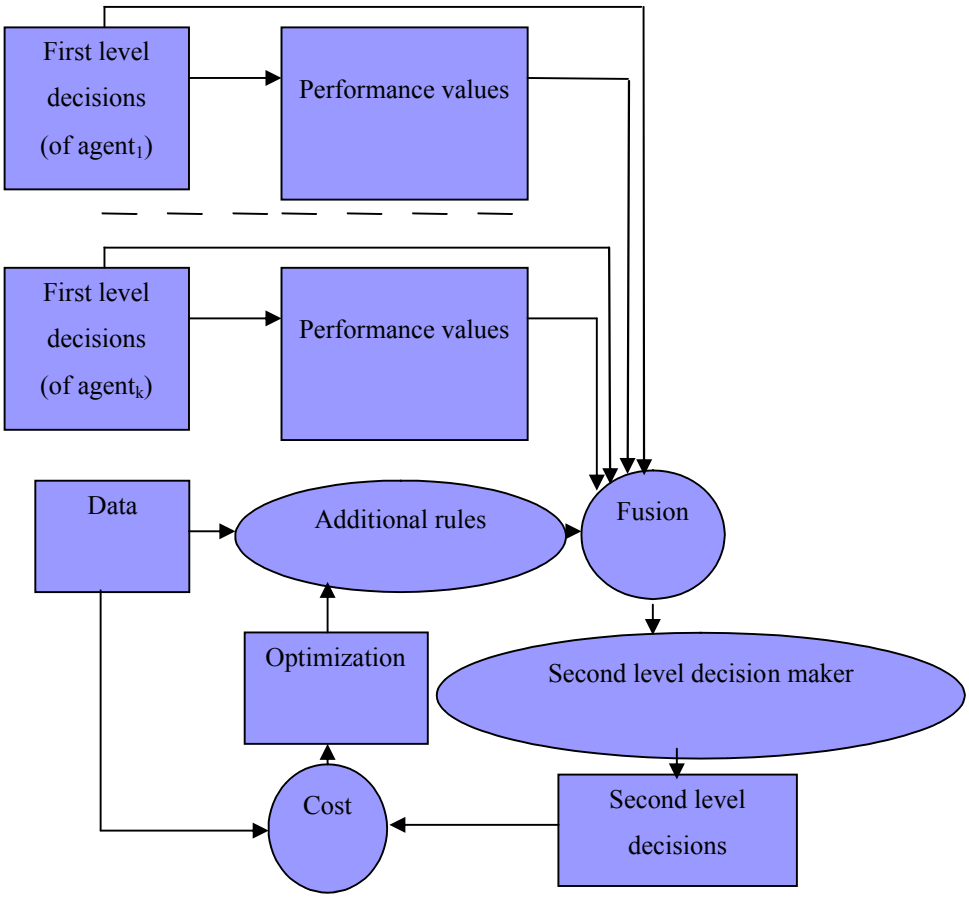

Fig. 1.4: The scheme for the development of the second level decision maker using a multi-agent structure in the first level.

The second multi-level DM model uses local agents at the levels (local DM in multiple-levels or LDM-ML). The main idea in this model is to develop agents that are supposed to work properly in restricted regions of the input search space and fuse the decisions of the agents in each level using a suitable decision-fusion method. The development of the agents in the local regions is provided using a suitable cost function emphasizing the locality information in its expression. If an agent is successful in its local region due to a local performance criterion, it is retained in the agent set for the next level. Otherwise the agent is eliminated from the model and not employed in the fusion anymore. The details of the model are as follows: at the first level, a group of local agents is developed which is trained to

perform local decisions. The cost function to train an agent accentuates the locality information of the region for which the agent is trained. Hence, the agents concentrate on making consistent decisions in and around their local regions after they are trained. The particular data to train a local agent is chosen from the data pool closest to the corresponding local region. After the agents are developed, a decision fusion method, using the locality information of the decision makers, determines the final decisions at the first level. A local performance criterion is set for the local agents; this criterion determines which agents are to be inserted into the next level and which agents are to be eliminated. If an agent is eliminated, a number of new local agents are developed in the sub-regions of the local region associated with the eliminated agent to take the place of the eliminated agent. Indeed, the process of elimination of unsuccessful agents and inserting a number of new agents in place of the eliminated agents is done to improve the performance of the model. This procedure increases the chances of the new substituted agents being successful since they function in the reduced sub-regions of the corresponding local regions. Decision fusion determines the overall decisions once again. Due to a similar performance criterion which is employed at the first level, some of the agents being used are also eliminated at the second level. This level-by-level procedure continues until a predefined success value is reached. An example scheme for such a relationship between two levels is shown in Fig. 1.4.

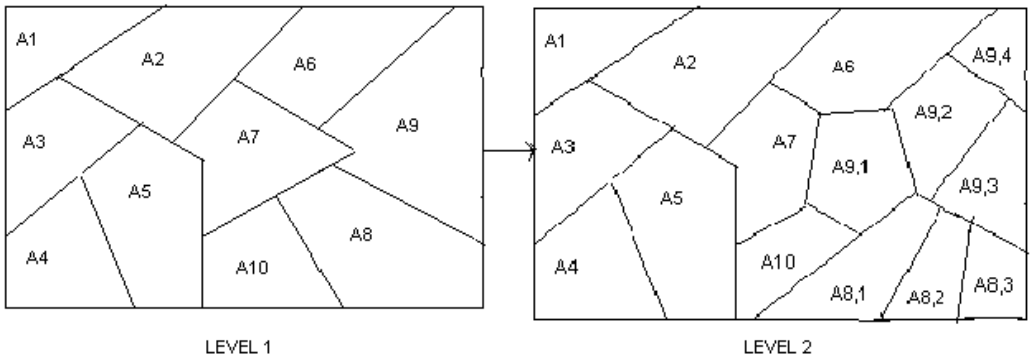

Fig. 1.5: The local agent configuration of two consecutive levels. At level 1, Agents 1-10 exist while at level 2, agents A8 and A9 are eliminated due to performance criterion, and in place of these agents, Agents A8,1 ; A8,2 ; A8,3 ; A9,1 ; A9,2 ; A9,3 ; A9,4 are inserted into the agent set. Because of the insertion of new agents, the regions of influence for some local agents have diminished (The regions influenced by agents A10, A6 and A7 decreased).

The summary of this thesis is that the basic functionality of the DM models depend on the utilization of previously taken decisions at lower levels 'partially or totally' at higher levels according to some performance measures over the lower level decisions. In order to relate the lower level decisions to higher level decisions, some intuitively intelligent decision-fusion techniques are proposed and employed which we call as "Multi-level DM models". Machine learning and optimization techniques are employed for training the experts in the models.

1.8. Advantages and Disadvantages of Proposed Decision-Making Models

There are three main advantageous of the proposed multi-level DM models. Firstly, they enable the construction of a decision maker for complex problems with satisfactory performance within a reasonable computation time. It is sometimes more advantageous to separate the problem and advise of a solution that requires optimization of a smaller number of variables than a giant structure that consumes indefinitely more computation time due to increased number of variables. Secondly, these models offer flexible structures. While applying the proposed DM model(s) to a problem, due to the requirement of policy changes in

problem solving, it is possible to easily use a variety of techniques or methods at different levels: Different agent structures can be used at various levels, the number of agents to use in a level can be changed, the functionality of the agents developed at each level can also be different. Various decision-fusion techniques can be applied depending on the required policy changes. The third important advantage of the proposed models is the utilization of the developed structures or their decisions at proceeding levels. This idea decreases the computation time and increases the robustness of the multi-level DM models.

1.9. Ensemble Classification and Fault-Detection Methods

In this thesis, a model fault-detection problem is used in order to examine the power of the proposed multi-level DM models. The models are used in order to classify the data into different sets associated with specified fault values in CS1. For this reason, a review of ensemble classification and fault detection concepts are investigated in order to picture the placement of the proposed multi-level DM models in this thesis with respect to other methods.

An effective method for classification is ensemble learning methods: Simpler base classifiers are developed whose decisions are aggregated via a suitable fusion technique (linear combination, averaging, majority voting) to achieve a final decision under the supervision of a superior classifier. Among ensemble learning methods, two of them are of much concern: Bagging [26] and Boosting (the Adaboost is the mostly used algorithm [27]).

In bagging, the original training set of instances is sampled to produce various training sets to develop base classifiers. The decision fusion for the ensemble classifiers are performed via an aggregation method like averaging or majority voting. Among the boosting algorithms, Adaboost is the main approach. It is similar to the concept of sequential DM. It is a base classifier combination approach depending on linear combinations of weak classifiers to produce a superior classifier [28]. It has many application areas, especially in face detection

[29, 30], target tracking [31], pattern recognition [32]. It is an iterative adaptive approach for updating the weights of poorly classified instances in the linear combination process to give them a greater chance in subsequent iteration steps. Initially a sequence of weak classifiers is produced. The relative difficulty of classification of each instance is demonstrated via a probability density function revealing the comparative resistance of the instance to being correctly classified. Initially this density function possesses a uniform distribution for all instances. At every iteration step, the relative importance of misclassified (or not so well classified) instances is increased and a variation in the density function for the instances occurs at the next iteration step. Updates of the parameters of the weak classifiers (the coefficients of linear combinations to produce a superior strong classifier) depend on the probability density function for each instance**.** There are several variants of the default Adaboost algorithm. Mainly, the adjusted algorithms are different from the default algorithm due to small parametric changes. Either a subset of all example samples is used in training or learning rates are taken in different styles [33]**.** Occasionally, neighborhood information between the instances is also inserted into the algorithm [34]**.** Parallelization of the algorithm is another typical application for better performance [35]. The Adaboost can be used with different machine learning techniques like support vector machines [36], or data evaluation and representation techniques like FL [37].

Some studies over the classification focus on the determination of the probability and cost assignments of the proposed algorithms. An attempt is observed for this issue in [38]. The requirement for developing learners that perform classification in a medium where unequal cost distribution for the instances is observed is studied in [38]. The study focuses on the techniques of generating cost sensitive learners for several classification algorithms.

Evidence theory is another tool used in the classification applications. [39] is about the fusion of decisions of ensemble decision makers by using evidence theory. Using Evidence theory has some advantages with respect to other fusion techniques which use different evaluation methods. In [40] the fusion of multi-

sensor data observations using evidence theory is the main concern. The data from different sensors may have imprecise and sometimes conflicting forms with each other. A mass function (probability function) is obtained using the distance between the faults and the measurements form the sensors. After the normalization process of the distances, the outputs of the process are used to construct the mass function. Total mass function is shaped using these normalized terms, which gives the probability assignment for a fault that occurs but the study makes two assumptions: The first assumption is that the faults are independent of each other and the second is that only one fault may occur at any one time. So, this structure can be used for detecting single fault scenarios.

There are several fault detection and diagnosis techniques available in the literature. Some of these techniques depend on controlling threshold values of related variables about the fault structure. More sophisticated data-driven techniques are observed in chemical or industrial processes depending on analysis of the related data. [41-42]. Also, principal component analysis, statistical methods, and Fisher discriminant analysis are used to solve problems related to the fault diagnosis approach. One of the main approaches used in fault detection is model-based approaches. The deviations between actual system response and observed system response which are obtained using mathematical models of these two systems are used for fault detection purposes. The fundamental idea in model-based fault detection is to create the error data between these two systems: The differences between actual system outputs and outputs of a representative system with faults might be a sign for a fault occurring in the process. The differences between these two observations are called as ‘residuals’. Residuals are potential symptoms for errors in the system. However, because of the noise or imperfections in the design of the model, the residuals might still be observed even if there is no fault (i.e., sign of a false fault). Different data evaluation techniques are applied for fault detection. [43] is about model-based fault detection method using FL tools. [44] is another study where FL is used in model-based fault detection processes. In [45], a locally recurrent NN structure which enables usage of past inputs as well as

current inputs, is designed in order to generate the residuals. The evaluation of residuals for the identification of faults is performed using statistical analysis.

Several hierarchical fault detection procedures can also be observed in the literature. For example, in [46] a hierarchical fault detection structure is proposed. Subsystem residuals are generated from the models of the subsystems of the main system. These residuals are transferred to residual evaluation units. Fault detection for the subsystems is separately evaluated in these units. Then, a supervisor fault detector determines the final decisions using the outputs of residual evaluation units. The hierarchy explained in this study is in terms of the separation of the main system into the subsystems and the generation and evaluation of associated residuals and the combination of their output knowledge in a higher level hierarchy. Another hierarchical fault detection application is [47]: A hierarchical Petri net model benefiting from FL tools as the inference mechanism is designed for fault diagnosis. The inference about an output in the first hierarchy level model is used in the inference for an output in the second hierarchy level model where each model has different structures and inputs. The hierarchy in this study is in terms of the relationships between the inputs and outputs of consecutive hierarchies.

The studies cited about fault detection mainly concern single fault structures. Finding the fault structure in multi-fault cases is a difficult task and not so many studies exist in the literature for this purpose. Another difficulty in fault detection is that a fault in some part of the system may create some residuals that alarm a fault in another part of the system; these are called as false faults. Isolation of false faults is important. Wrong alarms are important difficulties in fault detection. There is also another possibility: The faults may be dependent. In that case a fault in some part of the system may be observed as another fault in some other part. Dependent faults are some other complications that most of the methods utilized in fault detection generally hardly recover.

If the procedure used in the construction of CS1 is inspected, it can also be viewed as a model-based fault detection method. A fault-free model and a faulty model are used to generate residuals for the process and then these residual values are used for training in the development of the DM models. The basic advantage of the proposed DM models is the identification of specific faults for multi-fault scenarios.

1.10. Contributions of the Dissertation

The fundamental contributions of the dissertation can be summarized in the following items:

- Two new multi-level DM models have been suggested that can be applied for solving complex DM problems. The structures of these multi-level DM models are flexible and they allow for policy changes at different levels.
- The models have been applied on two case studies. The efficiencies of the proposed models are demonstrated for different model settings.
- The functionality and convergence of the models are demonstrated in two different proofs with some a priori assumptions about the models.
- The constructed models have been compared with some well-known algorithms: The advantages and disadvantages of the models compared to those of the algorithms have been demonstrated.
- CS1 has a complex nature. It is a fault-detection problem containing multi-fault situations. Proposed DM models have been used to find the fault amounts for more than one system parameter in general. Fault detection in multi-fault scenarios is an important issue where not many studies have been done until now.
- The development of multi-level DM models only requires the input-target relations. It is enough to know the circumstances that decisions are performed for (the inputs) and the correct decisions for the circumstances (the targets) to construct the models. For this reason, for any DM problem

whose input-target relations are available, it is possible to implement the proposed DM models with suitable additional modifications (the agent structures, the affairs between the agent structures).

- The main difference between the models proposed in this study and similar studies in literature is that, the proposed DM models enable and facilitate future development of the DM model even if decisions are unsatisfactory. The construction and the development and of the model is a continuous process due to the requirements in problem solving. The levels provide a facility to renew, update and enlarge the framework of the DM model. This facility exists for a few of the DM models in literature. Even if the proposed model does not work properly for a few levels, one can easily provide some modification for the affairs inside a level and between levels and let the model continue to deal with the DM problem in a specific and satisfactory manner. For most of the DM models in literature, if the model does not yield satisfactory solutions one should leave the model and start with a new model to solve the problem. But this is not a big difficulty for the proposed DM models in this thesis.

1.11. Presented Conference Papers and Accepted Journal Papers

During this dissertation, the following conference papers have been published:

a) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Hiyerarşik Karar Verme ve Karar Birleştirme,** SIU 2007 (15. Sinyal İşleme ve Uygulamaları Kurultayı), Eskişehir Anadolu University, June 11-13, 2007.

b) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Bulanık Mantık ile Hiyerarşik Karar Verme ve Karar Birleştirme**, TOK'07 Otomatik Kontrol Ulusal Toplantısı, Sabancı University, İstanbul, September 5–7, 2007**.**

c) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Developing Growing Hierarchical Structures for Decision Making,** ISCIS'07 **-** The 22nd International Symposium on Computer and Information Sciences, Middle East Technical University**,** Ankara, Turkey, November 7-9, 2007.

d) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Yerel Karar Verme ve Karar Birleştirme Tekniğinin Hata Bulmak Amacıyla Kullanılması ve Bayes Sınıflandırıcı ile Karşılaştırılması,** I. Mühendislik ve Teknoloji Sempozyumu, Çankaya Üniversitesi, Ankara, Turkey, April 24-25, 2008.

e) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Local Decision Making and Decision Fusion in Four-Tank Water Circulation System for Fault Detection (poster),** International Workshop on Operational Research (IWOR'08), Madrid, Spain, 5-7, June 2008.

f) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Evolving a Hierarchical Decision Making Mechanism Using Fuzzy Logic,** 17th IFAC (International Federation of Automatic Control) World Congress, Seoul, South Korea, July 6-11, 2008.

The accepted journal papers are as follows:

a) ***Ulaş Beldek,*** *Kemal Leblebicioğlu,* **Local Decision Making and Decision Fusion in Hierarchical Levels,** TOP**:** An Official Journal of the Spanish Society of Statistics and Operations Research**,** 17, 44-69, 2009.

Two more journal papers are planned to be submitted for publication in near future.

1.12. The Organization of the Chapters

The CS1 is explained in Chapter 2. HDM model is applied in Chapter 3 and Chapter 4. In Chapter 3 and 4, rule-bases made up of different representation structures and different evaluation techniques are used as the dynamic part of the agents developed at the hierarchical levels. One of the differences between the applications of Chapter 3 and 4 is the number of agents used at each hierarchical level. In Chapter 3, each hierarchical level of the applications contains a single agent whereas multiple agents are found at hierarchical levels of applications of Chapter 4. In Chapter 5, the LDM-ML model is applied to CS1. This chapter also

includes the Bayes optimal classifier applied for CS1. In Chapter 6, the Adaboost algorithm is applied to CS1. Chapter 7 includes the convergence analysis of the multi-level DM models. In Chapter 8 the multi-level DM models are applied to CS2. Finally, in Chapter 9 conclusive comments are presented.

CHAPTER 2

CASE STUDY 1: FAULT DETECTION IN FOUR-TANK WATER CIRCULATION SYSTEM

In this chapter CS1 on which the DM models will be constructed is explained. CS1 is a fault detection problem associated with a dynamical system. Using the mathematical model of the dynamical system, error data is constructed using the differences of state vectors corresponding to scenarios with faults (associated with parameter changes) and fault-free scenarios. This chapter is a preparation for the DM models that are explained and realized in the forthcoming chapters.

2.1. Defining the Problem

An artificial problem is set up in order to develop the multi-level DM models explained in Chapter 1. It is associated with a four-tank water circulation system. Several artificial failures are created at the bottom of the tanks that either allow extra water flow or diminish the amount of water flow through the tanks. The dynamics of the system are modeled by a set of non-linear differential equations. Using the mathematical model of the dynamical system, the error data is constructed using the differences of state vectors corresponding to faulty scenarios (associated with parameter changes) and fault-free condition. The error data coupled with the fault characteristics (amount of faults) is employed as input-target pairs to train the agents of the multi-level DM models. Agents are supposed to decide the fault amount for each sample.

The system has been shown in Fig. 2.1 below, with all the four tanks having holes at their bottoms. Consequently, some amount of water in the tanks flows through these holes. Water flows from upper tanks into the lower tanks as well as from the

lower tanks into the reservoir on the ground. There are two pumps to re-circulate the water that has aggregated in the reservoir back into the tanks

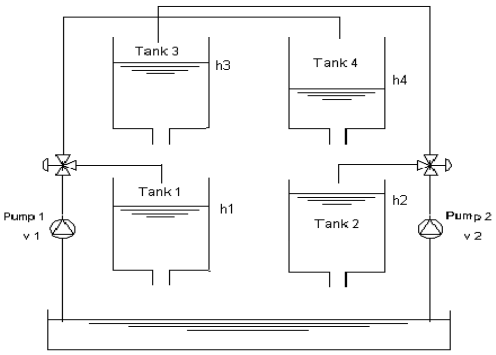

Fig. 2.1: Original configuration of four-tank system.

The holes at the bottom of the tanks have some nominal values. The dimensions of the holes are changed (increased or decreased) artificially to generate failures. The dynamics of the original multi-tank system and the dynamics of the multi-tank system with faults are presented in equations (2.1) and (2.2), respectively.

$$\begin{aligned}
\frac{dh_1}{dt} &= \frac{\gamma_1 k_1}{A_1} v_1 - \frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3}, \\
\frac{dh_2}{dt} &= \frac{\gamma_2 k_2}{A_2} v_2 - \frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4}, \\
\frac{dh_3}{dt} &= \frac{(1-\gamma_2) k_2}{A_3} v_2 - \frac{a_3}{A_3}\sqrt{2gh_3}, \\
\frac{dh_4}{dt} &= \frac{(1-\gamma_1) k_1}{A_4} v_1 - \frac{a_4}{A_4}\sqrt{2gh_4}.
\end{aligned} \tag{2.1}$$

$$
\begin{aligned}
\frac{dh_1}{dt} &= \frac{\gamma_1 k_1}{A_1} v_1 - \frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} - \frac{a_{leak1}}{A_1}\sqrt{2gh_1}, \\
\frac{dh_2}{dt} &= \frac{\gamma_2 k_2}{A_2} v_2 - \frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} - \frac{a_{leak2}}{A_2}\sqrt{2gh_2}, \\
\frac{dh_3}{dt} &= \frac{(1-\gamma_2) k_2}{A_3} v_2 - \frac{a_3}{A_3}\sqrt{2gh_3} - \frac{a_{leak3}}{A_3}\sqrt{2gh_3}, \\
\frac{dh_4}{dt} &= \frac{(1-\gamma_1) k_1}{A_4} v_1 - \frac{a_4}{A_4}\sqrt{2gh_4} - \frac{a_{leak4}}{A_4}\sqrt{2gh_4}.
\end{aligned}
\tag{2.2}
$$

In these equations, 'h_i' represents the height of the water in tank 'i'. The inputs to the system (the amount voltage applied to the pumps) are shown by 'v_1' and 'v_2'. The flow supplied by pump 'i' is '$\gamma_i v_i$'. 'A_i' and 'a_i' represent the cross-sectional areas of the tank 'i' and the outlet hole of tank 'i', respectively. 'γ_1' and 'γ_2' are the valve parameters and they determine how much water will be supplied to each tank from the pumps. These parameters are between 0 and 1. 'k_1' and 'k_2' are the pump constants. Consequently, the inflow supplied to tank 1, tank 2, tank 3 and tank 4 are '$\gamma_1 k_1 v_1$', '$\gamma_2 k_2 v_2$', '$(1-\gamma_2)k_2 v_2$' and '$(1-\gamma_1)k_1 v_1$', respectively. The parameter 'g' is the gravitational acceleration of the earth. The nominal values of these parameters are as follows:

- $A_1 = A_2 = A_3 = A_4 = 730$ cm^2.
- $a_1 = a_2 = a_3 = a_4 = 2.3$ cm^2.
- $v_1 = v_2 = 100$ Volt.
- $g = 981$ cm/sec^2.
- $k_1 = 5.51$ cm^3/Volt.sec.
- $k_2 = 6.58$ cm^3/Volt.sec.
- $\gamma_1 = 0.7$.
- $\gamma_2 = 0.6$.

In Equation (2.2), there are extra terms, (i.e., 'a_{leaki}') indicating the difference between the original system and the system with faults. These terms represent the deviations in the size of the holes (the faults). If the deviation is positive for hole

'i' (meaning that the area of the hole has increased with respect to the nominal size), 'a_{leaki}' is positive. If the deviation is negative for hole 'i' (meaning that the area of the hole has decreased with respect to the nominal size), 'a_{leaki}' is negative. The failure characteristic to be detected is the amount of fault (percentage deviation in the size of the hole) created in a specific tank for each time instant in the projected scenarios, where a scenario stands for the total fault conditions (single and multiple fault situations with their corresponding starting instants and duration) created in all the tanks (the nominal area of the tank holes are a_i = 2.3 cm^2).

Seven different fault levels are proposed. These are "Positive big deviation" (PB), "Positive medium deviation" (PM), "Positive small deviation" (PS), "No deviation" (Z), "Negative small deviation" (NS), "Negative medium deviation" (NM) and "Negative big deviation" (NB).

To develop agents in the DM model, 170 different training scenarios are generated. In each scenario, at least a single tank possesses a failure created at instant 't'. Each scenario takes place within 20 seconds and each fault is generated in the first 10 seconds. After it is created, the fault remains the same. In the training scenarios, for a particular tank 'i' at a particular instant 't', if the fault level is PB 'a_{leaki}' is assigned +2.3 cm^2, if the fault level is NB 'a_{leaki}' is assigned -2.3 cm^2, if the fault level is PM, 'a_{leaki}' is assigned as +1.53 cm^2, if the fault level is NM, 'a_{leaki}' is assigned -1.53 cm^2, if the fault level is PS, 'a_{leaki}' is assigned +0.76 cm^2 and if the fault level is NS, 'a_{leaki}' is assigned -0.76 cm^2 in Equation (2.2). Hence, the area of the hole in any tank ranges from 0 cm^2 to 4.6 cm^2 with the nominal hole size being 2.3 cm^2. The corresponding normalized hole deviations for the training scenarios are as follows:

- If the fault level is PB, normalized hole deviation is assigned +1.
- If the fault level is NB, normalized hole deviation is assigned -1.
- If the fault level is PM, normalized hole deviation is assigned +0.66.
- If the fault level is NM, normalized hole deviation is assigned -0.66.

- If the fault level is PS, normalized hole deviation is assigned +0.33.
- If the fault level is NS, normalized hole deviation is assigned -0.33.
- If there is no fault (Z), normalized hole deviation is 0.

Starting from the initial conditions ($h_1(0)$ = 20cm, $h_2(0)$ = 14cm, $h_3(0)$ =0 cm, $h_4(0)$ = 0 cm) Equation (2.2) is run for all the training scenarios for 20 seconds. At every 0.1 second, the water heights for the tanks are recorded for each scenario. Similarly, using the same initial conditions, Equation (2.1) is run once for 20 seconds. Then, the error data for the training scenarios is constructed as:

$$\begin{aligned} e_{1,scenario,i}(t) &= h_{1system\_with\_fault,scenario,i}(t)-h_{1original\_system}(t), \\ e_{2,scenario,i}(t) &= h_{2system\_with\_fault,scenario,i}(t)-h_{2original\_system}(t), \\ e_{3,scenario,i}(t) &= h_{3system\_with\_fault,scenario,i}(t)-h_{3original\_system}(t), \\ e_{4,scenario,i}(t) &= h_{4system\_with\_fault,scenario,i}(t)-h_{4original\_system}(t). \end{aligned} \tag{2.3}$$

In Equation (2.3), '$h_{k,original\_system}(t)$' represents the actual water heights measured for the tank 'k' employing Equation (2.1) at time instant 't', '$h_{ksystem\_with\_fault,scenario,i}(t)$' represents the water heights observed for the tank 'k' for the scenario 'i' at time instant 't' using Equation (2.2) with corresponding fault conditions for the concerned scenario, and '$e_{k,scenario,i}(t)$' represents the difference in the water height (error), calculated using these two observations for tank 'k' at scenario 'i' at time instant 't'.

The error data constitutes a 201×680 matrix where the 201 row represent the total time instants (from 0 to 20 seconds with 0.1 second time increments) and the 680 column represent the height differences for 4 tanks in 170 scenarios. Depending on the maximum and minimum values of height differences for each tank (i.e., e_1 e_2 e_3 and e_4 according to each scenario) the error data is normalized between -1 and 1. The configuration of the normalized error data (or briefly error data as used in the text in proceeding chapters) is given in Table 2.1.

Table 2.1: The configuration of error data (normalized error data).

Error Data	Scenario 1				…	Scenario 170			
Time Instants	e_1	e_2	e_3	e_4		e_1	e_2	e_3	e_4
1									
…									
201									

The error data samples and corresponding normalized fault amounts for each sample are used as the input-target pairs to train the agents to be employed in multi-level DM models. The configuration of the created fault characters for every training scenario is summarized in Table A.1.

In order to test the efficiency of the proposed multi-level models, two sets of validation scenarios are also designed. The first set of validation scenarios are created by varying the time instants that the faults are first initiated in the scenarios. The second set of validation scenarios are created by changing the faults in acceptable amounts ('a_{leaki}' values are varied: ±9% for both PS and NS faults, ±4.5% for both PM and NM faults, -3% or -6% for PB faults and +3% or +6% NB faults. The percentage changes for PS, NS, PM, NM faults and the 3% change both for PB and NB faults correspond to the same amount of hole area variations. On the other hand, the 6% percentage change for PB and NB faults correspond to twice the change in the hole area as compared to the previous cases. While choosing these percentage changes, the normalized fault amounts are taken into account. For instance, the related percentage change for a PS fault having a normalized fault amount of 0.33 results in a normalized fault amount of 0.3 or 0.36).

Construction of the error data for the validation scenarios is similar to the construction of error data for the training scenarios. However, the normalization procedure is performed according to the limits of the error data created for the training scenarios. The maximum and minimum values of the error data

corresponding to each tank for the training scenarios are used as the normalization factors. A second important point of concern is that the validation scenarios are planned such that none of the components of the samples of error data exceeds the associated maximum and minimum values obtained from training scenarios. Hence, it is certain that each component of error data will remain between 1 and -1 after normalization for the validation scenarios. The fault characters for these two sets of validation scenarios are shown in Table A.2 and Table A.3, respectively.

2.2. Noise

In order to test the effect of noise over the proposed multi-level models, two kinds of noise models with different distributions (uniform distribution and Gaussian distribution) are applied to the error data constructed for the training scenarios and the first set of the validation scenarios. Three different 'signal-to-noise ratios' (SNR) are chosen for each distribution (high, medium, and law). Formation of noisy error data (error data + noise) is done such that no component of the error data is noise-free (i.e., the noisy error data having a specific SNR value with a specific distribution).

Considering the noise model with uniform distribution,

- If the SNR is high, each noise component is formed using uniform distribution between 0.01 and -0.01.
- If the SNR is medium, each noise component is formed using uniform distribution between 0.05 and -0.05.
- If the SNR is low, each noise component is formed using uniform distribution between 0.1 and -0.1.

The noise is added to the error data both for the training and the validation scenarios. However, some limitations are proposed such that none of the components of noisy error data exceeds the limit values 1 or -1 (if a component of

a sample of noisy error data exceeds the limit values, it is reassigned to these limit values).

In the noise model with Gaussian distribution,

- If the SNR is high, each noise component is created using Gaussian distribution with 0 mean and 0.006 variance.
- If the SNR is medium, each noise component is created using Gaussian distribution with 0 mean and 0.028 variance.
- If the SNR is low, each noise component is created using Gaussian distribution with 0 mean and 0.06 variance.

The noise having Gaussian distribution is added to the error data both for the training and the validation scenarios concerning the same principle as uniform noise (no component of a sample of noisy error data exceeds the limit values -1 or 1).

After generating the noisy error data with different settings of uniform and Gaussian distribution, their SNR's are calculated as follows:

- The power of noise-free error signal is calculated: The squares of the components of each noise-free sample are summed up to calculate the total noise-free error signal power.
- The power of noise signal is calculated: The squares of each noise component are summed up to calculate the total noise signal power.
- SNR is calculated by dividing the square root of noise-free error signal power to the square root of noise signal power.

The SNR's of noisy error data created for the training scenarios and the first set of validation scenarios with different SNR settings of the uniform distribution and the Gaussian distribution are demonstrated in Tables 2.2 and 2.3, respectively.

Table 2.2: SNR's for uniform distribution

SNR	High	Medium	Law
For noisy error data created using the training scenarios	45.7834	9.1779	4.5912
For noisy error data created using the validation scenarios	45.5298	9.1079	4.5558

Table 2.3: SNR's for Gaussian distribution

SNR	High	Medium	Law
For noisy error data created using the training scenarios	44.0047	9.4414	4.4212
For noisy error data created using the validation scenarios	43.6577	9.3674	4.3858

2.3. Remarks

The error data obtained for the training scenarios are used in order to develop the agents that are used in the multi-level DM models. The error data constructed for both of the validation scenarios and the noisy error data created, both for the training scenarios and the first set of validation scenarios are used to test the developed multi-level models in terms of robustness to environmental changes and resilience to noise, respectively. Similarly, the error data with these different construction styles are also employed for comparison purposes with well-known available techniques (the Adaboost and the Bayes optimal classifier) in the forthcoming chapters.

CHAPTER 3

HIERARCHICAL DECISION-MAKING MODEL (RULE-BASE STRUCTURE 1)

In this chapter, the HDM model proposed in Chapter 1, is applied to the CS1. GA is employed to develop the agents in the form of rule-bases that perform the decisions in the hierarchical levels. FL is utilized as the interpolation technique for combining the rules. Briefly, the model is realized as follows: A first level decision maker (agent), which is a rule-base, is developed to produce the first level decisions. These decisions are forwarded to the second level with a measure of reliability determined by a performance criterion, associated with the success of the first level decisions at different local decision regions. Then, a new second level decision maker is developed by the help of the first level decision maker (i.e., benefiting from the decision of the first level decision maker) in a similar way at the second level.

The decision maker at the second level comprised of two different entities. The first entity (static part) is the first level decisions and their reliability values whereas the second entity (dynamic part) is the new rule-base to be trained in the second level. The decisions of the new rule-base are combined with the first level decisions based on a decision fusion algorithm. This combination yields the raw second level decisions. An optimization process determines the rules of the new rule-base. Eventually, the second level decisions are obtained. Thus, the first level decisions help development of the second level decision maker. The relation between two consecutive levels is similar to the relation between the second level and the first level.

In the applications of this chapter, the dynamic part of the decision makers are rules made up of '**IF .. THEN ..**' statements. Combination of these rules is made

by the ‘min-max’ type FL interpolation. GA based optimization is used to construct the rules in the dynamic parts.

There are many different ways to apply the HDM model to the CS1. The performance criterion can be defined differently; various decision fusion algorithms and different rule-base structures can be utilized. The effects of some of these choices on the performance of the overall DM model have been investigated. The performance of a standard single level direct (non-hierarchical) approach is also compared with the results obtained using HDM model in different applications. In the applications where HDM model are employed, a single decision maker is developed at each level. The fusion equation based on the reliability values of lower level decisions, is a weighted convex averaging operation of the lower level decisions and decisions of new rule-base being developed.

3.1. Applications

Seven different applications have been carried out to check the effects of different choices. Among these applications, the first two are performed to determine the extent the rule structures can be augmented. These two applications give a clue about the optimal number of premises in the rules.

The applications 3 to 6 are performed in order to analyze the effects of different performance criteria. For each application (Each application includes 5 or 6 hierarchical levels), the decision fusion is performed based on different performance criterion and their effects in the training and the validation scenarios are observed. Eventually, the efficiency of the agents at noisy environments is monitored in application 6 which demonstrates seemingly the best result in training.

The last application (application 7) is performed in order to compare the results obtained using HDM model with the results obtained using a non-hierarchical

model which consist only of a single rule-base. In this application the rule-base possesses a more sophisticated (and intuitively more powerful) structure compared to the rule-bases that are employed in HDM model in previous applications. The comparison is made in terms of the computation time and training performance.

3.1.1. Application 1

In the first application, different rule-base structures are exploited at each level. In lower levels, the rules of rule-base are simpler whereas in the upper levels the rules seem to be more sophisticated.

3.1.1.1. The First Level

The aim in the first level is to develop a simple rule-base that performs the first level decisions. GA is used as the optimization technique for development: Each rule-base in the population of a GA generation is coded as a chromosome. All the rule-bases are made up of 30 '**IF...THEN..'** statements (rules). Each '**IF...THEN..'** statement in the rule-base with the premise and consequent variables are concatenated to each other constructing the structure of a chromosome. The chromosomes (coded rule-bases) have the duty to perform the decisions for each input: They predict the normalized fault amounts for every instant at each scenario in the selected tank. Each instance of error data corresponds to an input related with an instant of a scenario. These inputs are used to train the rule-base: The rule-base uses FL as the interpolation technique to combine the rules. The obtained output is the normalized fault prediction corresponding to the input. The cost function to be used in the optimization (using GA) process is composed of the sum of the absolute difference between the real normalized fault amount and the predicted normalized fault amount for every input. The rule-base obtained at the end of the optimization process is declared as the first level agent. The agent's decisions (predictions) are used in the

construction of the second level agent. The structure of a rule in a rule-base at the first level is as below:

IF ($e_{1,t}$='att_1' AND $e_{2,t}$='att_2' AND $e_{3,t}$='att_3' AND $e_{4,t}$= 'att_4') THEN (p_t= 'att_5')

In this rule structure, '$e_{1,t}$', '$e_{2,t}$', '$e_{3,t}$' and '$e_{4,t}$' are the premise variables. They represent the normalized water height differences for each tank at instant 't'. 'p_t' is the consequent variable. It reflects the conclusion of the rule at instant 't'. The premise variables '$e_{1,t}$', '$e_{2,t}$', '$e_{3,t}$' and '$e_{4,t}$' can be assigned eight different attributes. These attributes are 'negative-big', 'negative-medium', 'negative-small', 'zero', 'positive-small', 'positive-medium', 'positive-big' and 'not important'. Different from the premise variables the consequent variable can be assigned seven different attributes excluding only 'not important' attribute. The membership functions for the premise variable $e_{1,t}$ are shown in Fig. 3.1. The membership functions for the remaining variables (both premise variables and consequent variable) are similar to Fig. 3.1, as well.

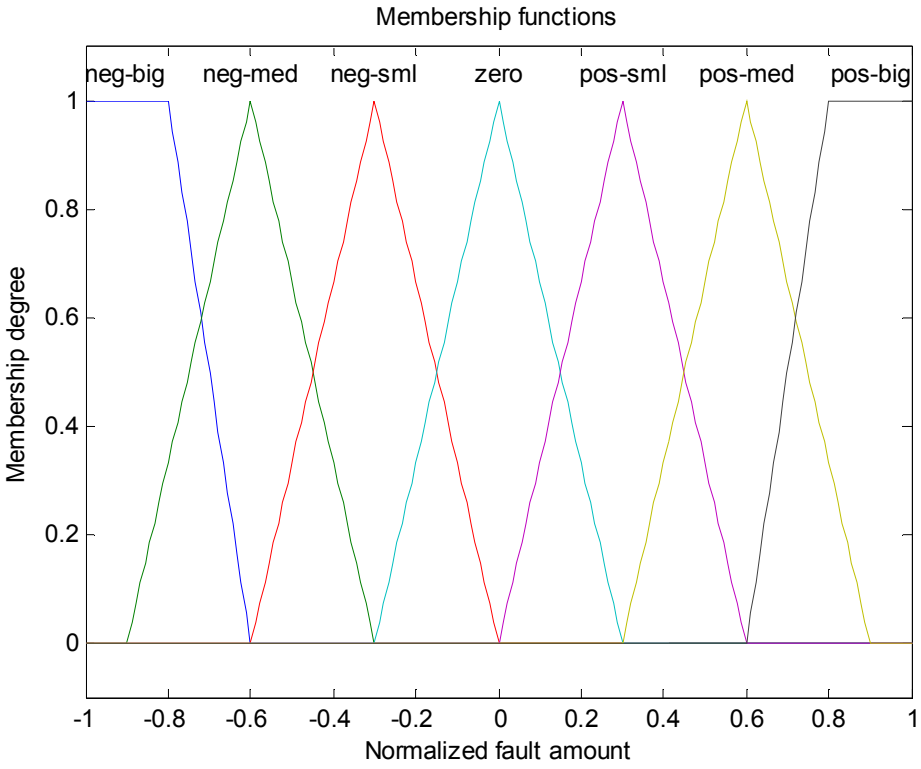

Fig. 3.1: **Membership functions of each attribute for the variable $e_{1,t}$: 'neg-big' for negative big 'neg-med' for negative-medium, 'neg-sml' for negative-small, 'zero' for zero, 'pos-sml' for positive-small, 'pos-med' for positive medium, 'pos-big' for positive big**

The reason that trapezoid membership functions are used for negative-big and positive-big attributes is to equalize the effects of these attributes with other attributes:

In the rule structure, 'AND' operates as the 'minimum' operation: Using the membership functions, each premise variable is assigned a membership degree corresponding to an input. 'AND' operation evaluates the minimum of the membership degrees assigned to the premise variables. The output (of 'AND' operation) is assigned as the membership degree of the consequent variable. A simple illustrative example for use of 'AND' is shown in Fig. 3.2.

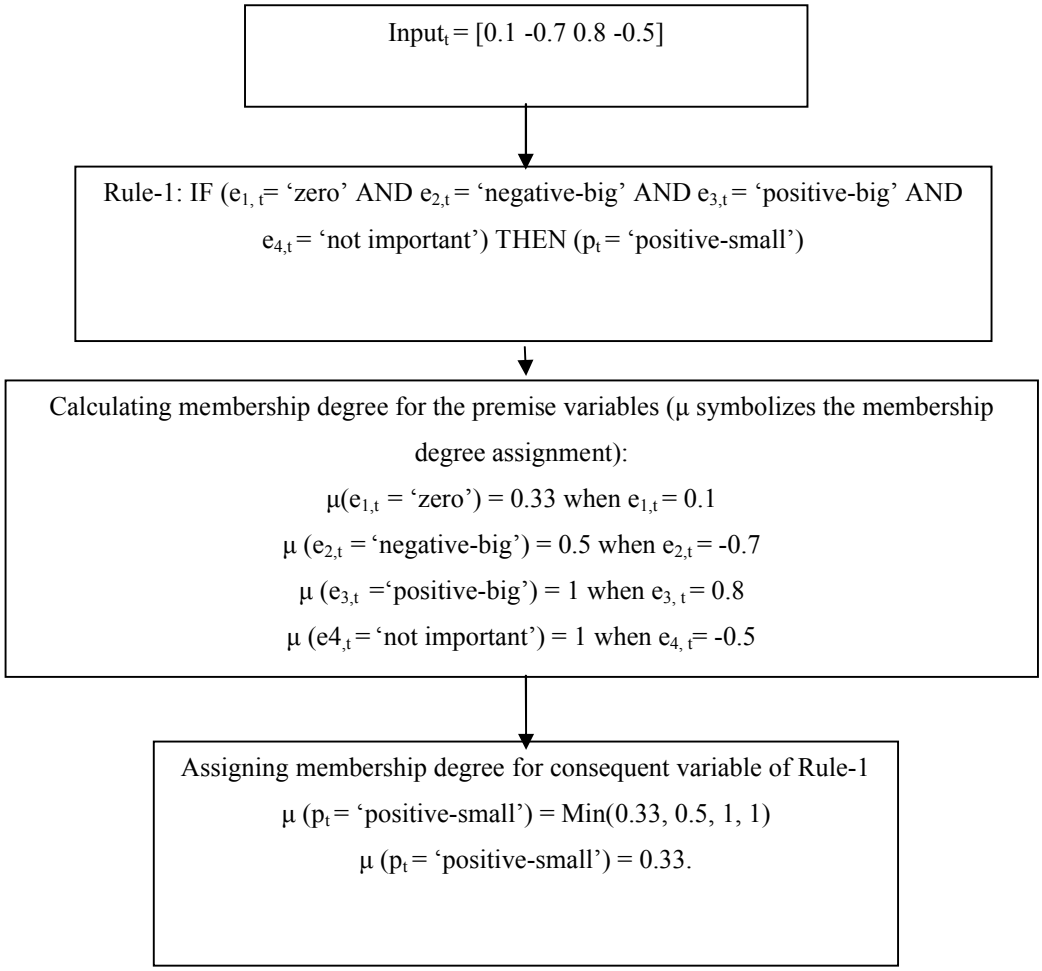

Fig. 3.2: Example: The membership degree assignment for the consequent variable of a rule due to input at time 't'.

If a premise variable is assigned the attribute 'not important', it becomes the ineffective element for 'AND' operation. In other words, the membership degree for the premise variable is assigned as 1.

The aggregation of rules' conclusions (the membership degrees of consequent variables of each rule in the rule-base) is realized through a 'maximum' operation: For each different attribute (excluding 'not-important') a membership degree is assigned as follows:

- For each rule in the body of the rule-base, the membership degrees for the consequent variables are determined.
- If same attribute is assigned to the consequent variables of two or more rules, the maximum of the membership degrees evaluated for the consequent variables of corresponding rules is assigned as the membership degree of the corresponding attribute.
- If a rule possesses an attribute at its consequent variable different from all the attributes assigned to the consequent variables of all other rules, the membership degree assigned for the consequent variable for the rule is assigned as the membership degree of the attribute.
- If no rule contains a specific attribute assigned to its consequent variable, the membership degree for that specific attribute is assigned to 0.

The output of the aggregation operation yields attributes with membership degrees. An illustrative example for the 'maximum' operation is shown in Table 3.1.

The membership degrees assigned to each attribute determine an output surface with the help of the corresponding membership functions: The membership degree assigned to an attribute shows the maximum value that related membership function can be appointed. The area under the membership function curve restricted by the membership function degree is the portion of the membership function to be considered in the construction of the output surface for the corresponding attribute. The same procedure is performed for each attribute and eventually the construction of the output surface is completed as the union of computed areas for each attribute. The surface obtained at the end of the 'maximum' operation for the rule-base shown in Table 3.1 is given in Fig. 3.3.

Table 3.1: Example for determination of membership degrees of all the 7 attributes (positive-big, …, negative-big) for a rule-base having 7 rules.

Rules of the rule-base		
Rule no:	Attribute of consequent variable	Membership degree of consequent variable
1	Negative-big	0.35
2	Positive-small	0.20
3	Negative-big	0.05
4	Positive-small	0.15
5	Zero	0.10
6	Positive-small	0.05
7	Zero	0.10

Assigned membership values for each different attribute	
Attribute	Membership degree
Negative-big	max(0.35,0.05) = 0.35
Negative-medium	0 (inactive)
Negative-small	0 (inactive)
Zero	max(0.10,0.10) = 0.10
Positive-small	max(0.20,0.15,0.05) = 0.20
Positive-medium	0 (inactive)
Positive-big	0 (inactive)

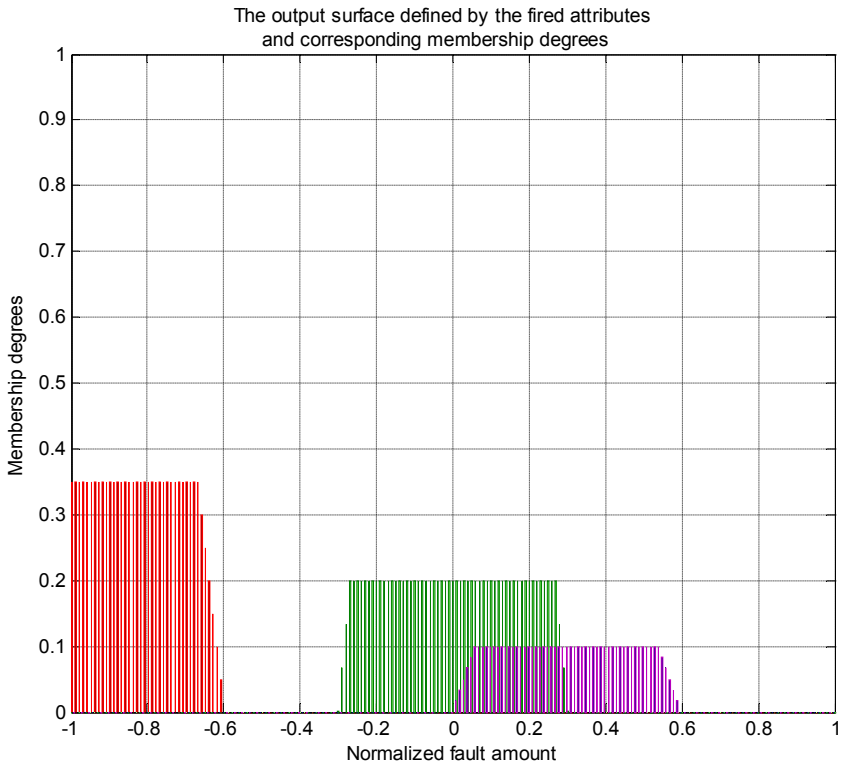

Fig. 3.3: **Fired attributes and corresponding membership degrees of resultant output surface created for the example demonstrated in Table 3.1.**

The centre of area of the output surface is declared as the normalized first level fault amount predicted for handled input. The centre of area defuzzification is summarized as:

$$n\_f\_f\_p = \frac{\sum_i \left(\int_{-1}^{1} x f_i(x)dx\right)}{\sum_i \left(\int_{-1}^{1} f_i(x)dx\right)}, \tag{3.1}$$

where, '$f_i(x)$' symbolizes the equation of the membership function limited up to its membership degree for the attribute 'i' and 'n_f_f_p' symbolizes the normalized first level fault amount prediction corresponding to an input.

The prediction is a value between -1 and 1. It shows the predicted amount of the normalized change from the nominal value in the hole of the tank. An example is produced in the following Table 3.2.

Table 3.2: Explanation of predictions.

Prediction	Explanation
0	There is no change in the hole size (0% change)
1	The hole size is doubled (100% positive change in hole size)
-1	The hole is totally closed (100% negative change in hole size)
0.5	The hole size is increased 50% with respect to the nominal hole size

The sum of absolute differences between the predicted normalized fault amounts and real normalized fault amounts for all the inputs yields the cost of a chromosome as

$$Cost = \sum_{k=1}^{170} \sum_{t=1}^{201} \left| n\_f\_f\_p_t(k) - n\_r\_f_t(k) \right|, \tag{3.2}$$

where, 'n_f_f_p_t(k)' represents the normalized first level fault prediction at the instant 't' in the k^{th} scenario based on the chromosome, 'n_r_f_t(k)' represents the normalized real fault value at instant 't' in the k^{th} scenario. There are totally 34170 samples to be considered in the cost function construction (170 scenario × 201 time instants for each scenario). The fitness of a chromosome is obtained by taking the reciprocal of the cost.

In a rule, there are 5 variables (4 premise variables and 1 consequent variable). A rule-base enclosing 30 rules possesses 150 variables, totally. Hence a chromosome is made up of 150 genes. The attributes these variables are assigned are optimized in the GA simulation. The parameters and the results for the first level GA simulation for Tank 1 are as follows:

- The number of chromosomes in a population: 40.
- Number of generations: 2000.

- The number of genes in a chromosome: 150.
- Crossover operation: Within each consecutive 25 genes, one point crossover is performed.
- Crossover ratio: 90%.
- Reproduction: 10 % (with elitist method).
- Gene mutation probability: 1% (The mutation operation is only applied on half of the chromosomes undergoing crossover operation, and it is applied after the 10^{th} generation. If the fitness of the best chromosome remains the same for 8 consecutive generations the mutation rate is increased to 5% for the next generation. Mutation rate is returned to its nominal value (1%) in the following generations.
- The fitness of the best chromosome: 2.5160×10^{-4}.
- The cost of the best chromosome: 3974.
- Fault amount per sample: 3974/34170 = 0.1163.

The fitness value of best chromosome at each generation is shown in Fig. 3.4.

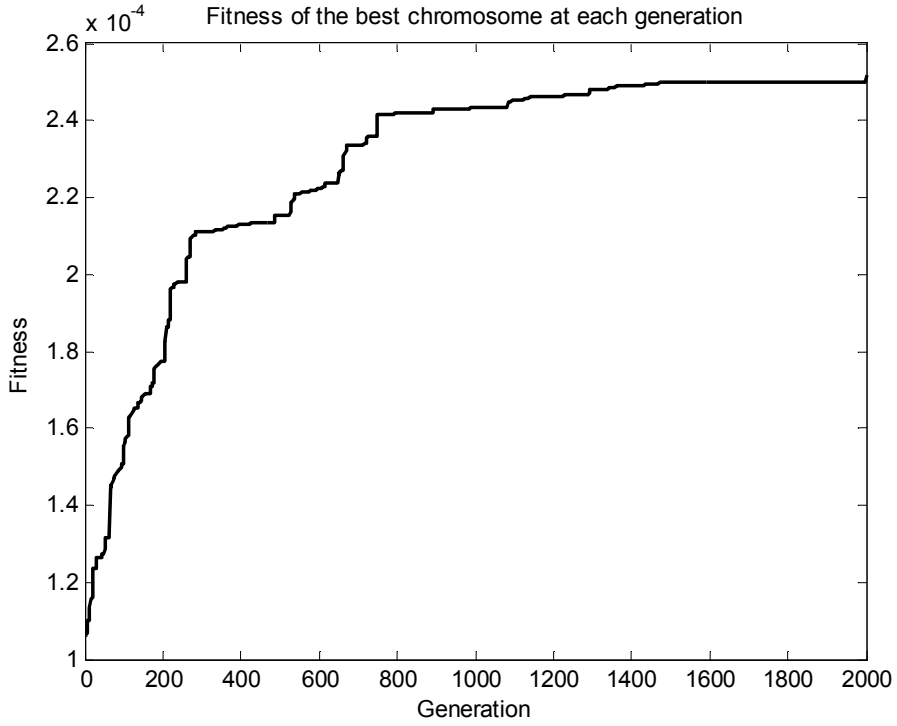

Fig. 3.4: Fitness of the best chromosome at each generation (application 1, the first level).

The average fitness value of each population at each generation is shown in Fig. 3.5.

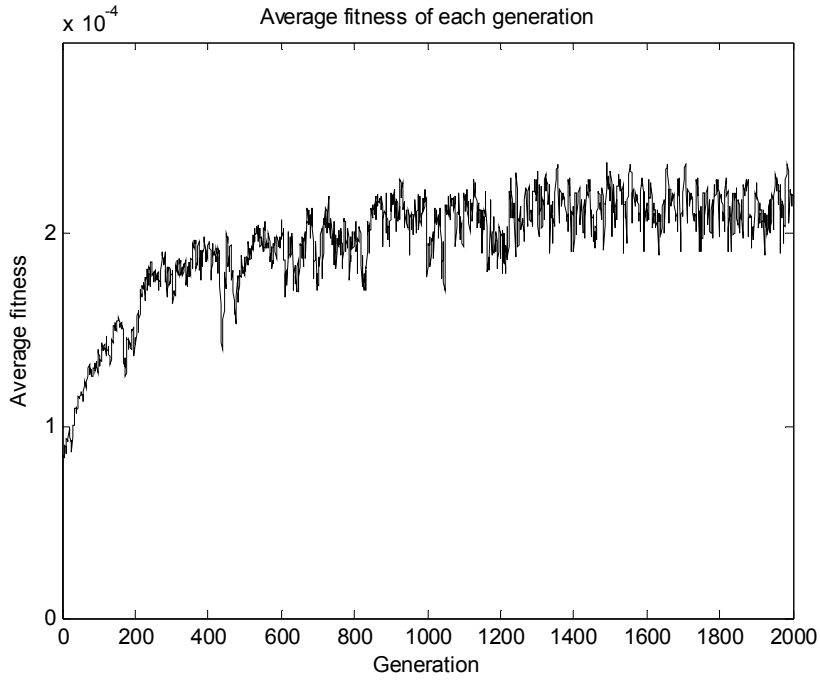

Fig. 3.5: The average fitness of each generation (application 1, the first level).

In the second level simulations this cost function decreases more using the HDM model.

3.1.1.2. The Second Level

The second level decisions are obtained with the help of the first level decisions: The main idea is to combine the decisions obtained at the first level with the decisions of a newly developing rule-base to create the second level primitive decisions. An optimization process is carried out to train the new rule-base by minimizing a cost function concerning the second level decisions. The fusion technique is simple. The first level decisions are transferred to the second level with their reliability values. At the second level, the decisions of the first level agent and the decision of a newly developing rule-base are combined using a fusion equation employing the reliability values for the first level decisions to construct the second level decisions. Second level agent is constructed after the

additional rules are determined using an optimization process. Thus, the second level agent can be explained as the unification of two parts: One part is the static part coming from the first level decisions and their reliability values. The other part is dynamic part which is composed of the newly developing rule-base. The fusion of the first level decisions and the decisions of newly developing rule-base is a convex averaging operation employing from the reliability values of the first level decisions in different decision regions.

The dynamic part of the second level decision maker is a new rule-base: It employs two consecutive samples of error data corresponding to the instants 't' and 't-1' to produce the normalized fault amount prediction for the instant 't'. The rule-base uses the same techniques as in level 1 to yield its output for a given input (same membership degree assignment principles, same membership functions for all the variables). The rule structure can be briefly summarized as follows:

IF ($e_{1,t}$='att_1' AND $e_{2,t}$='att_2' AND $e_{3,t}$='att_3' AND $e_{4,t}$='att_4' AND $e_{1,t-1}$='att_5' AND $e_{2,t-1}$='att_6' AND $e_{3,t-1}$='att_7' AND $e_{4,t-1}$='att_8') THEN p_t='att_9'

In this rule structure '$e_{1,t}$', '$e_{2,t}$', '$e_{3,t}$' and '$e_{4,t}$' are the first set of premise variables. They represent the normalized water height differences at each tank for instant 't'. '$e_{1,t-1}$', '$e_{2,t-1}$', '$e_{3,t-1}$' and '$e_{4,t-1}$' are the second set of premise variables. They represent the normalized water height differences at each tank for instant 't-1'. 'p_t' stands for the consequent variable representing the partial conclusion of the rule at the time instant 't'.

The fusion of the decisions of the first level agent and the decisions of newly developing rule-base is accomplished via the help of a performance criterion demonstrating the capability of the first level agent in performing consistent decisions at different decision regions. The performance criterion and its representation (i.e., the performance plot) are obtained as follows:

Step 1- For each normalized first level decision, determine its decision region and a representative point inside this decision region. The points of concern are chosen either as the mid points or the boundary points of the decision regions.

- If -1 ≤ prediction < -0.833, then decision region = negative-big, the representative point for the decision region is -1.
- If -0.833 ≤ prediction < -0.5, then decision region = negative-medium, the representative point for the decision region is -0.66.
- If -0.5 ≤ prediction < -0.166, then decision region = negative-small, the representative point for the decision region is -0.33.
- If -0.166 ≤ prediction ≤ 0.166, then decision region = zero, the representative point for the decision region is 0.
- If 0.166 < prediction ≤ 0.5, then decision region = positive-small, the representative point for the decision region is 0.33.
- If 0.5 < prediction ≤ 0.833, then decision region = positive-medium, the representative point for the decision region is 0.66.
- If 0.833 < prediction ≤ 1, then decision region = positive-big, the representative point for the decision region is 1.

Step 2- For each decision region; determine the sum of the absolute difference between the real normalized fault amounts and predicted fault amounts belonging to the decision region.

Step 3: Divide the sums to the number of decisions belonging to a decision region to find out the error amount per sample for each decision region.

Step 4- Error amounts are subtracted from 1 to obtain the approximate success rates for each decision region. Mathematically, the procedure above can be summarized as

$$s\_r_i = 1 - \frac{\sum_{n\_f\_f\_p_j \in dec\_reg_i} \left| n\_r\_f_j - n\_f\_f\_p_j \right|}{num(n\_f\_f\_p_j \in dec\_reg_i)}. \qquad (3.3)$$

In Equation (3.3), 's_r_i' defines the success rate for the decision region 'i', '$n\_r\_f_j$' is the real normalized fault amount at the instant 'j', '$n\_f\_f\_p_j$' is the corresponding decision at the instant 'j'. '$n\_f\_f\_p_j \epsilon dec\_reg_i$' shows the normalized first level fault decisions belonging to decision region 'i', and 'num(.)' operation

returns the number of sample. There are several ways to obtain success rates. Indeed the real equation should be

$$s\_r_i = 1 - \frac{\sum_{n\_f\_f\_p_j \in dec\_reg_i} \left| n\_r\_f_j - n\_f\_f\_p_j \right|}{2 \times num(n\_f\_f\_p_j \in dec\_reg_i)}. \quad (3.4)$$

where the term '2' is inserted into denominator since the difference between a predicted normalized fault amount and real normalized fault value could be 2 maximally (obtained when prediction is 1 and real fault amount is -1 or vice versa). Otherwise, it is possible that success rates are negative. However, an agent developed for the first level is used in order to calculate the success rates, and it is observed using Equation (3.3) that it is possible to obtain success rates greater than 0. There is another reason to use Equation (3.3) to calculate the success rates: It is observed that when Equation (3.4) is used, the success rates become so high for the first level agent that (each of the 's_r_i' values become very close to 1), due to the fusion function little emphasis will be given to newly developing rule-base in the second level. For these two reasons, Equation (3.3) is preferred to calculate the success rates instead of Equation (3.4).

Step 5- After success rates are calculated, the values of success rates are assigned to the representative points for the corresponding decision regions. Eventually, by a spline interpolation these success rates and their representative points are combined and performance plot is obtained.

The performance plot is a continuous function showing approximate reliability of any decision made. In Fig. 3.6, the performance plot for the first level decisions are shown.

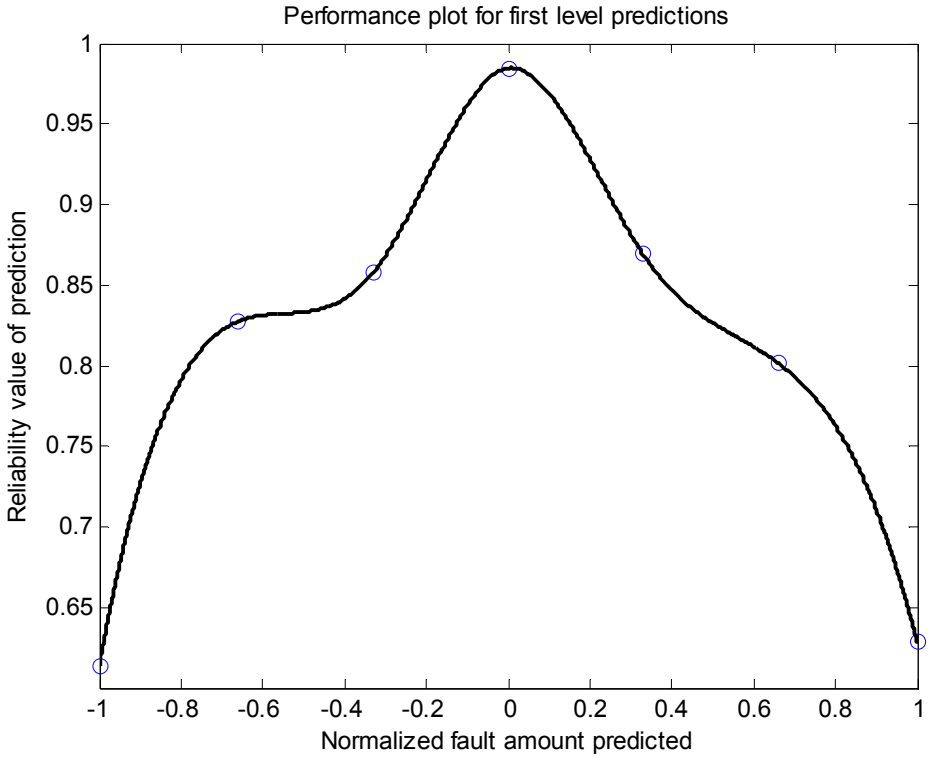

Fig. 3.6: The performance plot showing the reliability values of the first level predictions (application 1).

From the performance plot, the reliability values of any prediction (decision) could be easily computed. For example, prediction of '0' (meaning there is no fault in the tank) has a reliability value between 0.95 and 1. As another example, prediction of 0.2 has a reliability value between 0.9 and 0.95.

The fusion of the first level prediction with the prediction of the new rule-base to produce the second level prediction (i.e., fusion equation) can be written as

$$slp = (flp \times rel\_flp) + (nrbp \times (1 - rel\_flp)), \qquad (3.5)$$

where, 'slp' is the second level prediction, 'flp' is the first level prediction, 'rel_flp' reliability value of the first level prediction; 'nrbp' is the prediction of new rule-base. The fusion equation is a weighted convex summation depending on the reliability value of the first level prediction. If the first level prediction is

comparatively weak in some decision region (i.e., its reliability value is small), much freedom is granted to the newly developing rule-base to adjust the second level prediction resulting from the weight term '(1-rel_flp)'. The second level prediction with its dynamic (new rule-base) and static (the first level prediction and the reliability value) parts is an integral part of optimization process. The cost function for a chromosome at the second level is,

$$Cost = \sum_{k=1}^{170} \sum_{t=1}^{201} \left| slp_t(k) - real\_nor\_fault_t(k) \right|. \qquad (3.6)$$

where '$slp_t(k)$' represents the second level prediction performed for the time instant 't' for the 'k^{th}' scenario, '$real\_nor\_fault_t(k)$' represents the real normalized fault value. The fitness function is the reciprocal of the cost function.

The new rule-base in the second level is made up of 10 rules. A rule in the rule-base possesses 8 premise variables and 1 consequent variable (A chromosome should contain 90 genes (10× (8+1)). The second level simulation has the same GA parameters as the first level simulation. The results obtained for Tank1 are as follows:

Fitness of the best chromosome at the end of simulation: 2.621×10^{-4}.

- Cost of the best chromosome at the end of simulation: 3815.
- Amount of fault per sample: 3815/34170 = 0.1116.
- The improvement in cost with respect to the first level simulation: (|3974-3815|/3974) ×100 = % 4.

The fitness value of the best chromosome at each generation is shown in Fig. 3.7.

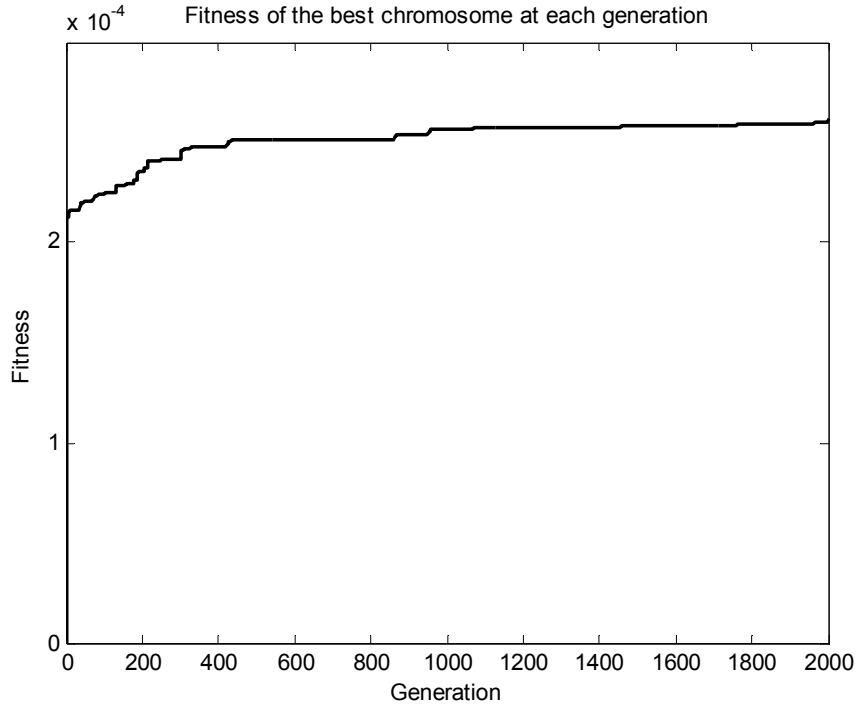

Fig. 3.7: Fitness of the best chromosome at each generation (application 1, the second level).

It is observed that there is an improvement in the fitness values when the second level results are compared with the first level results. This is expected: The decisions of the first level are transferred to the second level with reliability values showing the approximate success of the first level decisions at different decision regions and these decisions are used to construct the second level decisions by the help of an optimization process. It is also possible to increase the performance using new decision fusion techniques and new rule-base structures.

3.1.2. Application 2

The aim of the second application is to investigate the variation in the construction style of the performance plot and rule structure to see their effect in performance (fitness, cost, error per data).

3.1.2.1. The First Level

The first level of Application 2 is the replica of the first level of Application 1. So the results (developed agent structure) of the first level of Application 1 are directly transferred to the second application.

3.1.2.2. The Second Level

The only difference in the second level compared to the first application is how the success rates are determined. They are determined as follows:

Step 1- For each sample of error data; determine its attribute class according to its real normalized fault amount and assign a representative point for the attribute class similarly as in application 1:

- If $-1 \leq$ real normalized fault amount < -0.833, then attribute class = negative-big, the representative point for the attribute class is -1.
- If $-0.833 \leq$ real normalized fault amount < -0.5, then attribute class = negative-medium, the representative point for the attribute class is -0.66.
- If $-0.5 \leq$ real normalized fault amount < -0.166, then attribute class = negative-small, the representative point for the attribute class is -0.33.
- If $-0.166 \leq$ real normalized fault amount ≤ 0.166, then attribute class = zero, the representative point for the attribute class is 0.
- If $0.166 <$ real normalized fault amount ≤ 0.5, then attribute class = positive-small, the representative point for the attribute class is 0.33.
- If $0.5 <$ real normalized fault amount ≤ 0.833, then attribute class = positive-medium, the representative point for the attribute class is 0.66.
- If $0.833 <$ real normalized fault amount ≤ 1, then attribute class = positive-big, the representative point for the attribute class is 1.

Step 2- For each attribute class (negative-big, .., positive-big) possessing different ranges of real normalized fault amounts of the first level decisions, the sum of absolute differences between the real normalized fault amount and the predicted fault amount are calculated.

Step 2: The sums are divided by the number of samples belonging to the corresponding attribute class to calculate the error per sample for the corresponding class.

Step 3- Error per samples are subtracted from 1 to obtain the approximate success rates for each attribute class. Mathematically, the procedure above can be summarized as

$$s\_r_i = 1 - \frac{\sum_{n\_r\_f_j \in att_i} \left| n\_r\_f_j - n\_f\_f\_p_j \right|}{num(n\_r\_f_j \in att_i)}, \tag{3.7}$$

where 's_r_i' defines the success rates for different attribute classes 'i', '$n\_r\_f_j$' is the normalized fault amount for the sample 'j', '$n\_f\_f\_p_j$' is the corresponding decision for the sample 'j', '$n\_r\_f_j \epsilon att_i$' is the normalized real faults belonging to attribute class 'i' and 'num()' operation turns the number of samples.

After success rates are calculated, the values of success rates are assigned to the representative points for the corresponding attribute classes. Eventually, by a spline interpolation these success rates and their representative points are combined and the performance plot is obtained.

The performance plot obtained at the end of spline interpolation for the first level decisions is shown in Fig. 3.8.

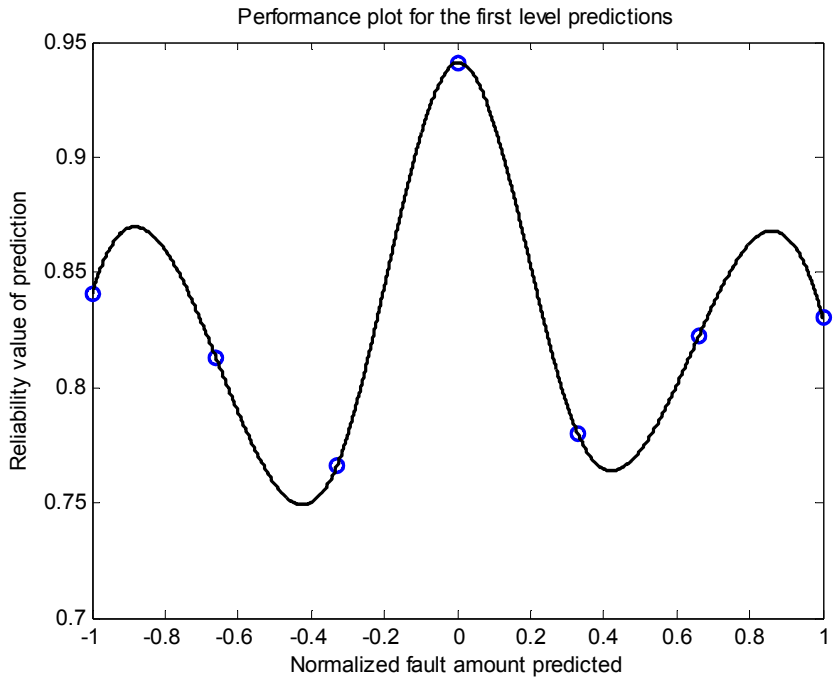

Fig. 3.8: The performance plot showing the reliability values of the first level predictions (application 2).

The parameters and the results of the second level are as follows: The new rule structure has 8 premise variables and 1 consequent variable. The rule-base is made up of 10 rules (A chromosome enclosing a rule-base should have 90 genes (10× (8+1)). The second level has the same GA parameters as the first level simulation. The results for Tank1 are as follows:

Fitness of the best chromosome at the end of simulation: 2.634×10^{-4}.

- Cost of the best chromosome at the end of simulation: 3797
- Amount of fault per sample: 3797/34170 = 0.1111.
- The improvement in cost with respect to the first level simulation: (|3974-3797|/3974) ×100 = % 4.4.

The fitness value of the best chromosome at each generation is shown in Fig. 3.9.

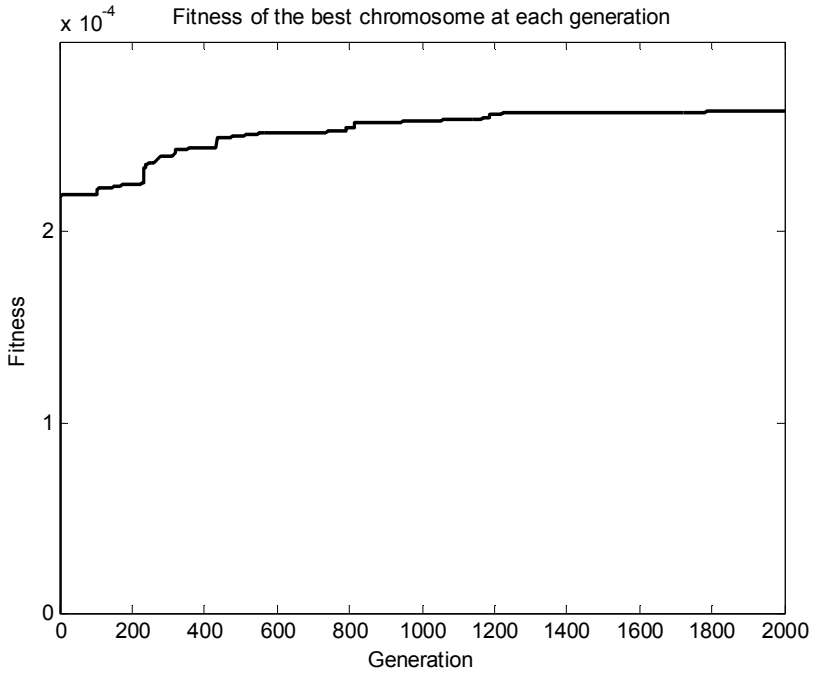

Fig. 3.9: Fitness of the best chromosome each generation (application 2, the second level).

As seen, using another performance plot construction style, an improvement is observed again.

3.1.2.3. The Third Level

A new rule structure which is more sophisticated is introduced in this level. It has the ability to gather and employ information from three consecutive instants.

IF ($e_{1,t}$='att_1' AND $e_{2,t}$='att_2' AND $e_{3,t}$='att_3' AND $e_{4,t}$='att_4' AND $e_{1,t-1}$='att_5' AND $e_{2,t-1}$='att_6' AND $e_{3,t-1}$='att_7' AND $e_{4,t-1}$='att_8' AND $e_{1,t-2}$='att_9' AND $e_{2,t-2}$='att_{10}' AND $e_{3,t-2}$='att_{11}' AND $e_{4,t-2}$='att_{12}') THEN p_t='att_{13}'

In this rule structure '$e_{1,t}$', '$e_{2,t}$', '$e_{3,t}$' and '$e_{4,t}$' represent the first set of premise variables. They represent the normalized water height differences for each tank at instant 't'. '$e_{1,t-1}$', '$e_{2,t-1}$', '$e_{3,t-1}$' and '$e_{4,t-1}$' represent the second set of premise

variables. They represent the normalized water height differences for each tank at instant 't-1'. '$e_{1,t-2}$', '$e_{2,t-2}$', '$e_{3,t-2}$' and '$e_{4,t-2}$' represent the third set of premise variables. They represent the normalized water height differences for each tank at instant 't-2'. 'p_t' stands for the consequent variable representing the partial conclusion of the rule at the instant 't'. The determination of the output for an input to the rule-base is the similar to the one in the first level (same membership degree assignment procedure, same membership functions for all the variables). The success rates are obtained using Equation (3.7). The representative points are same as in Section 3.1.2.2. The performance plot is constructed similarly as in Section 3.1.2.2. The fusion idea is similar to the one shown in Equation (3.5). However, the second level decisions and the reliability values deduced from the performance plot for the second level decisions are used to obtain the third level decisions. The cost of a chromosome is calculated using a function similar to Equation (3.6). However, the third level decisions are inserted into cost in place of the second level decisions.

Two different simulations are carried out to see the effect of an expanded rule structure (i.e., having some additional variables). The difference in the simulations is the sizes of the rule-bases (number of rules in the rule-bases).

The first simulation parameters and results are as follows: The new rule structure has 12 premise variables and 1 consequent variable. The rule-base is made up of 5 rules (A chromosome enclosing a rule-base should have 65 genes (5× (12+1)). The third level has the same GA parameters as the first level simulation. The obtained results for Tank1 are given below.

- Fitness of the best chromosome at the end of simulation: 2.5227×10^{-4}.
- Cost of the best chromosome at the end of simulation: 3964.1
- Amount of fault per sample: 3964.1/34170 = 0.1160.
- The **deterioration** in cost with respect to the second level simulation: (|3964.1-3797|/3797) ×100 = % 4.4.

The fitness value of the best chromosome at each generation is shown in Fig. 3.10.

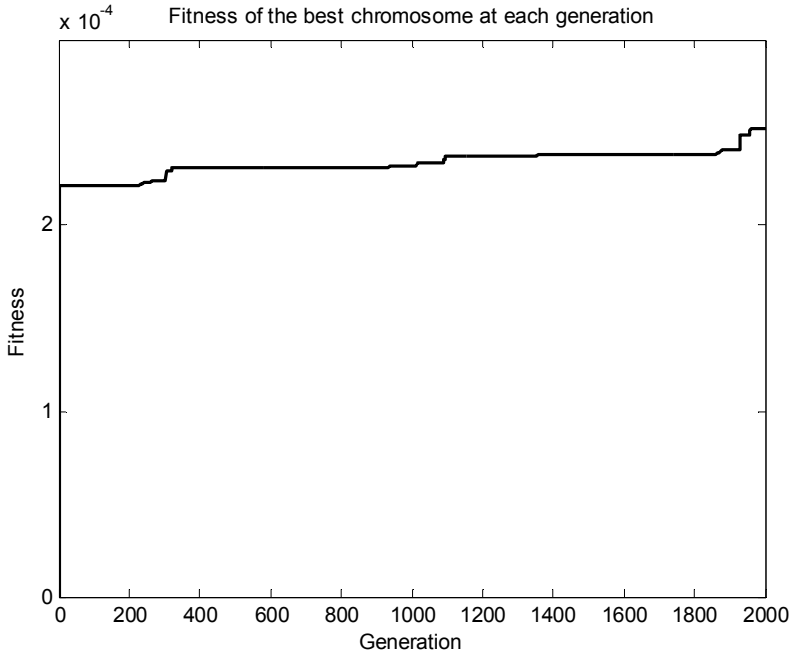

Fig. 3.10: Fitness of the best chromosome at each generation (application 2, the third level, 5 rules in a rule-base).

The second simulation parameters and results are as follows: The new rule structure has 12 premise variables and 1 consequent variable. The rule-base is made up of 10 rules (A chromosome enclosing a rule-base should have 130 genes (10× (12+1)). The third level has the same GA parameters as the first level simulation. The obtained results for Tank1 are as follows:

- Fitness of the best chromosome at the end of simulation: 2.5831×10^{-4}.
- Cost of the best chromosome at the end of simulation: 3871.3
- Amount of fault per sample: 3871.3/34170 = 0.1132.
- The **deterioration** in cost with respect to the second level simulation: (|3871.3-3797|/3871.3) ×100 = % 1.9.

The fitness value of the best chromosome at each generation is shown in Fig. 3.11.

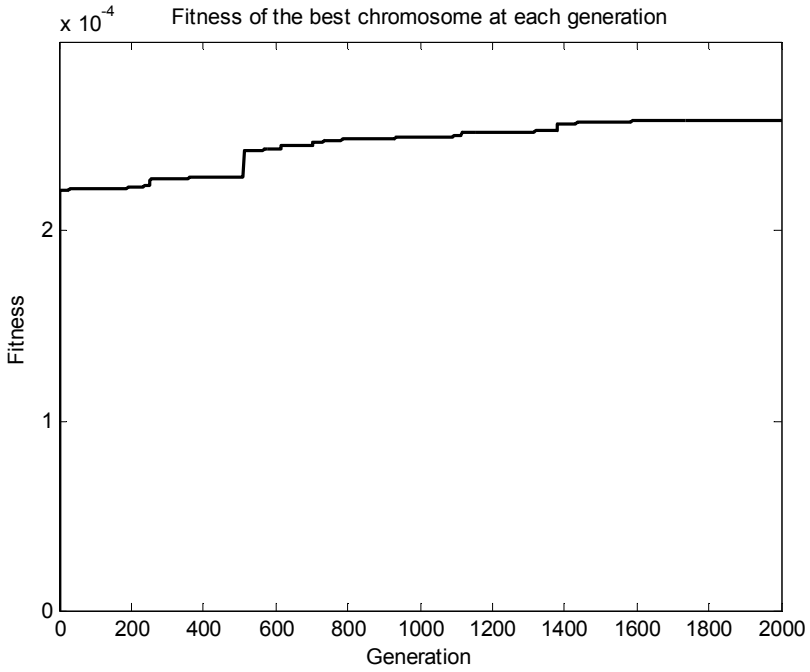

Fig. 3.11: Fitness of the best chromosome at each generation (application 2, the third level, 10 rules in a rule-base).

As seen from the results of the third level in both of the simulations, expanding the rule structure (i.e., increasing the number of variables in a rule) does not always bring improvement. There may be several reasons for this unsatisfactory result: The rule structure has become so complex that optimization of such a rule-base requires too much time. Increasing the variables in a rule may influence the relations between the rules (i.e., due to min-max type data interpolation technique) negatively. Another issue is about the dimensions of the rule-base: The number of rules in the rule-base may be insufficient to process the error data satisfactorily. However if the number of rules in the rule-base are further increased, the time spent to develop such a rule-base will also increase.

What could be done in order to increase the efficiency of the agents at consecutive hierarchical levels? One idea is to use the rate of change of error data with raw error data together. A new rule structure employing both information sources (i.e., error data and rate of change of error data) may increase the performance. Some

additional variables related with the rate of change of error data can be inserted into the rule structure in place of the variables related with error data at consecutive time instants (i.e., instants ‘t-1’ and ‘t-2’). However, as the number of premise variables in the rule structures is increased, development process becomes impossible. For these reasons inserting the rate of change of the error data is not a primary requirement. Hence, smaller rule structures are more suitable for our HDM model.

3.1.3. Application 3

In this application, Equation (3.3) is used to obtain the success rates and the performance plots at each level. HDM is carried out for 5 levels. The rule structure at each level is the same as the one used in the first level of previous applications. In the first level the rule-base is composed of 30 rules while in other levels each rule-base is made up of 20 rules.

For the first level the GA parameters are the same as the first level of the previous applications. But for higher levels the gene mutation probability is adjusted as follows: Gene mutation probability: 1% (The mutation operation is only applied on half of the chromosomes undergoing crossover operation, and it is applied after the 100^{th} generation. If the fitness of the best chromosome remains the same for 8 consecutive generations the mutation rate is increased to 5% for the next generation. Mutation rate is returned to its nominal value (1%) in the following generations). The results obtained are shown in Table 3.3.

Table 3.3: The results for the third application.

	The first level	The second level	The third level	The fourth level	The fifth level
Fitness	2.5160×10^{-4}	2.6855×10^{-4}	2.7764×10^{-4}	2.8549×10^{-4}	2.9319×10^{-4}
Cost	3974	3723	3601	3502	3410

Figs. 3.12, 3.13, 3.14 and 3.15 show the fitness of the best chromosome at each generation for the second, the third, the fourth, and the fifth levels, respectively.

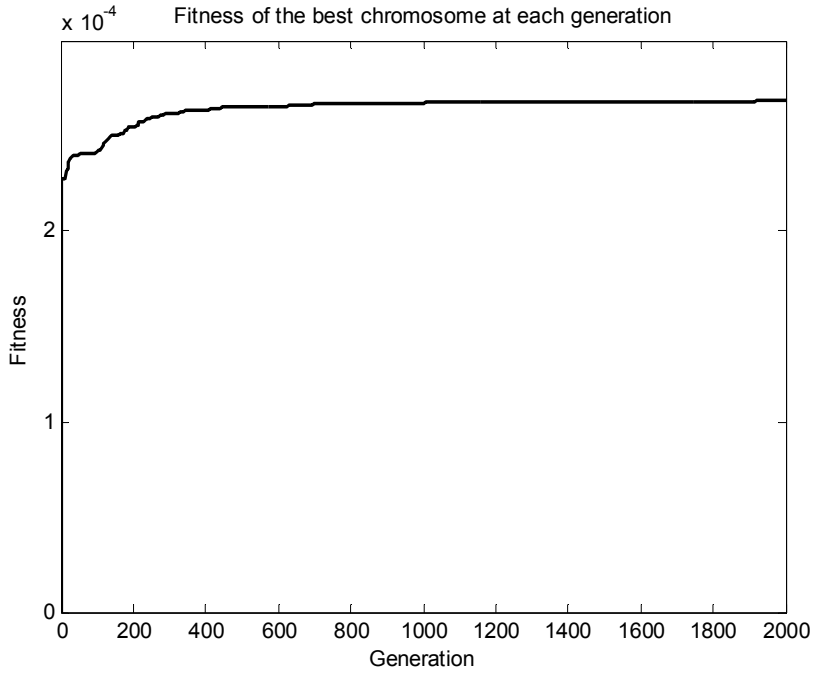

Fig. 3.12: Fitness of the best chromosome at each generation (application 3, the second level).

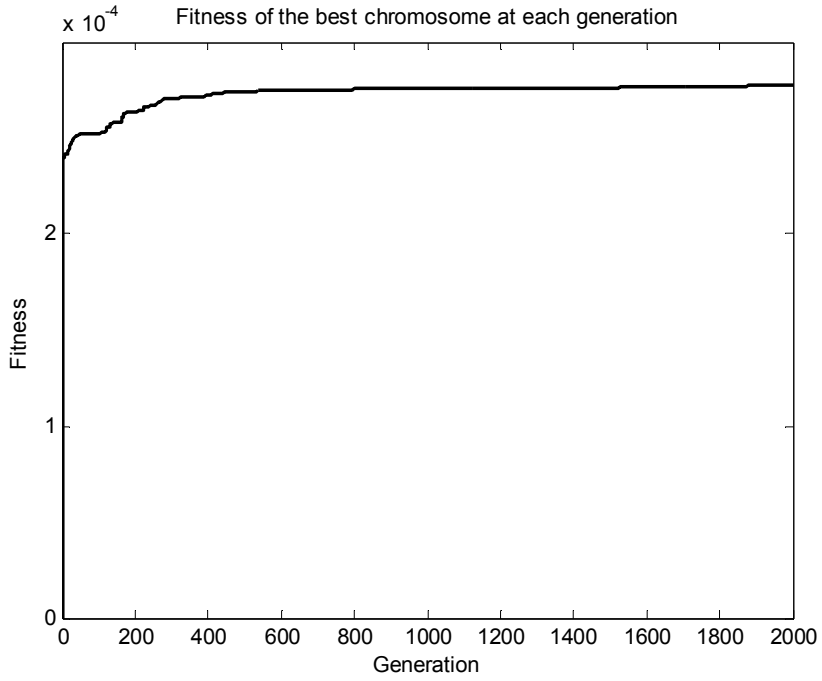

Fig. 3.13: Fitness of the best chromosome at each generation (application 3, the third level).

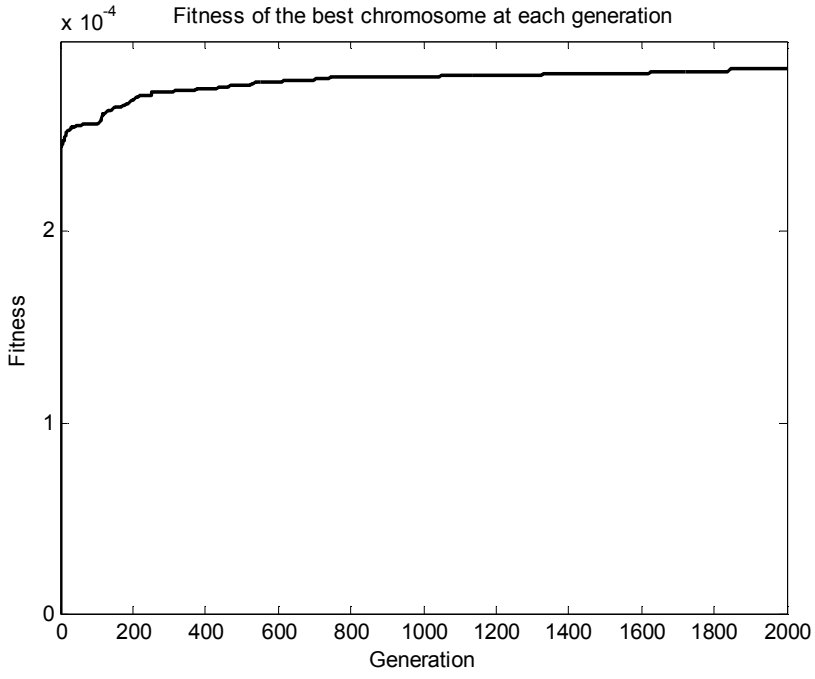

Fig. 3.14: Fitness of the best chromosome at each generation (application 3, the fourth level).

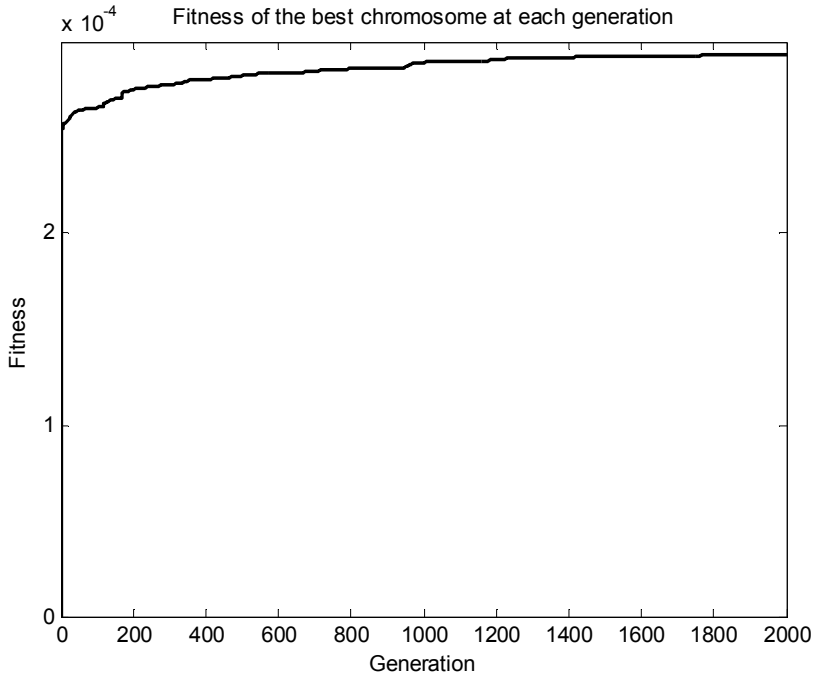

Fig. 3.15: Fitness of the best chromosome at each generation (application 3, the fifth level).

In Fig. 3.16, the performance plots related to the first, the second, the third, the fourth and the fifth levels are shown.

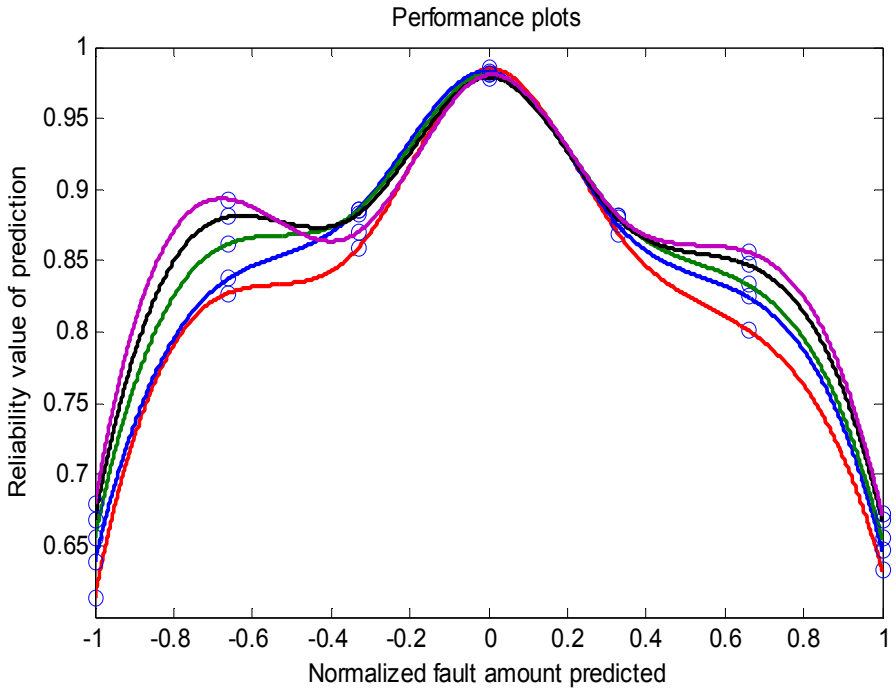

Fig. 3.16: Performance plots for the best agent at each level (application 3): Red for the first level, blue for the second level, green for the third level, black for the fourth level, violet for the fifth level.

After the agents of each level are created, their efficiency are tested in the validation scenarios (In the validation scenarios, the performance plots obtained for the training scenarios are used as the performance functions. It should be assumed that the normalized fault amounts for the validation scenarios are unknown before using the HDM model. The same approach should also be used in the noisy applications.). The results are as in the Tables 3.4 and 3.5:

Table 3.4: The results for the third application (for the first set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level
Fitness	2.4181×10^{-4}	2.5278×10^{-4}	2.5463×10^{-4}	2.6045×10^{-4}	2.7055×10^{-4}
Cost	4135	3956	3927	3839	3696

Table 3.5: The results for the third application (for the second set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level
Fitness	2.6521	2.8282	2.9224	3.0129	3.0877
Cost	3770	3535	3421	3319	3238

3.1.4. Application 4

In this application, Equation (3.7) is used to obtain the success rates and the performance plots are sketched accordingly at each level. The HDM model is carried up to 5 levels. The rule structure at each level is the one used in the first level of previous applications. In the first level the rule-base is composed of 30 rules, while in other levels each rule-base is made up of 20 rules. The GA parameters at each level are the same as the corresponding levels of application 3. The results obtained are shown in Table 3.6.

Table 3.6: The results for the fourth application.

	The first level	The second level	The third level	The fourth level	The fifth level
Fitness	2.5160×10^{-4}	2.6164×10^{-4}	2.8376×10^{-4}	2.9202×10^{-4}	2.9759×10^{-4}
Cost	3974	3822	3524	3424	3.360

Figs. 3.17, 3.18, 3.19, and 3.20 show the fitness of the best chromosomes at each generation for the second, the third, the fourth, and the fifth levels, respectively.

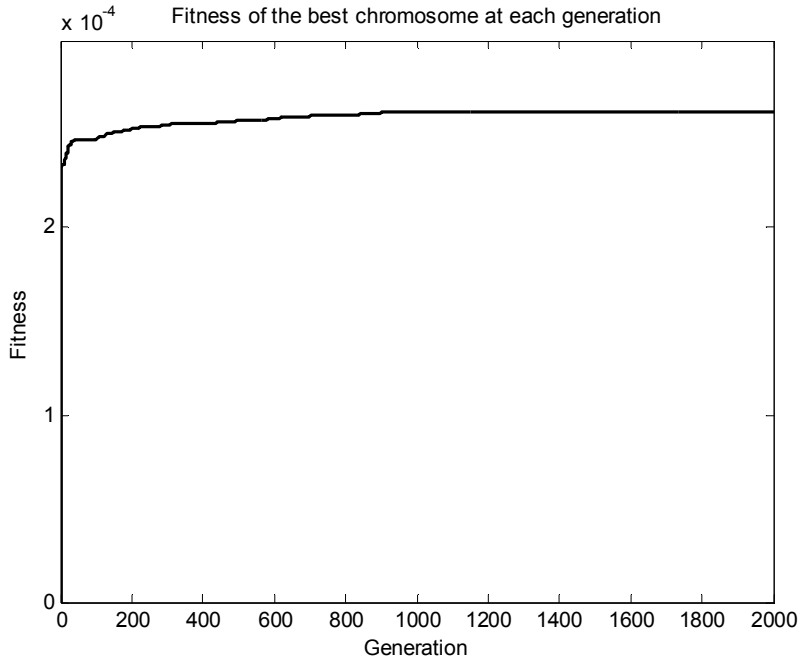

Fig. 3.17: Fitness of the best chromosome at each generation (application 4, the second level).

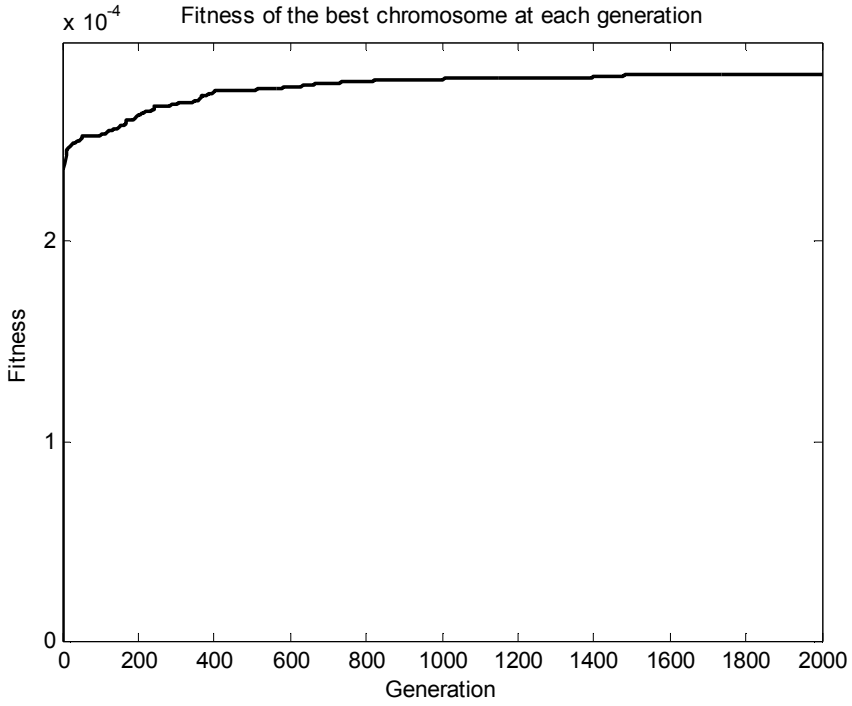

Fig. 3.18: Fitness of the best chromosome at each generation (application 4, the third level).

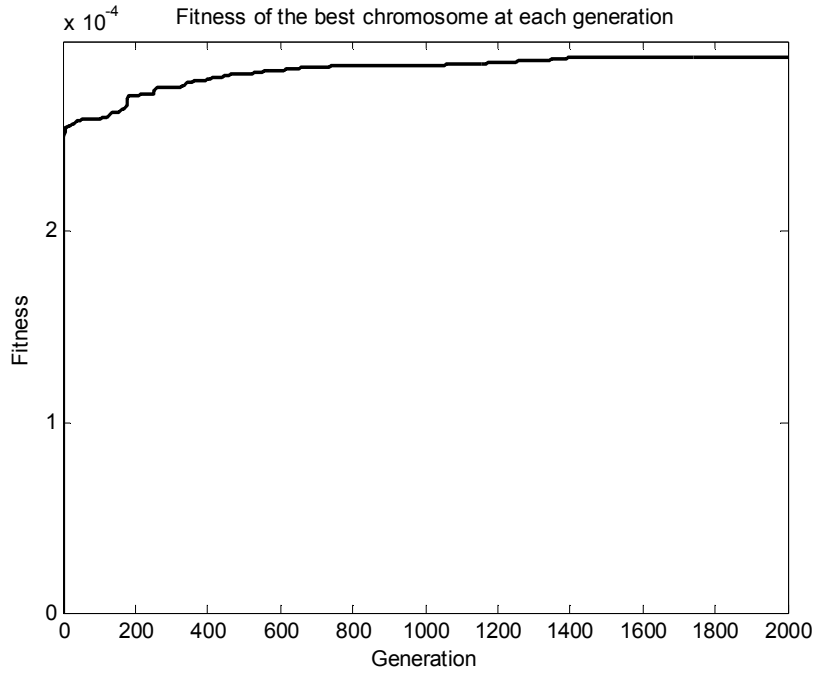

Fig. 3.19: Fitness of the best chromosome at each generation for the fourth level (application 4).

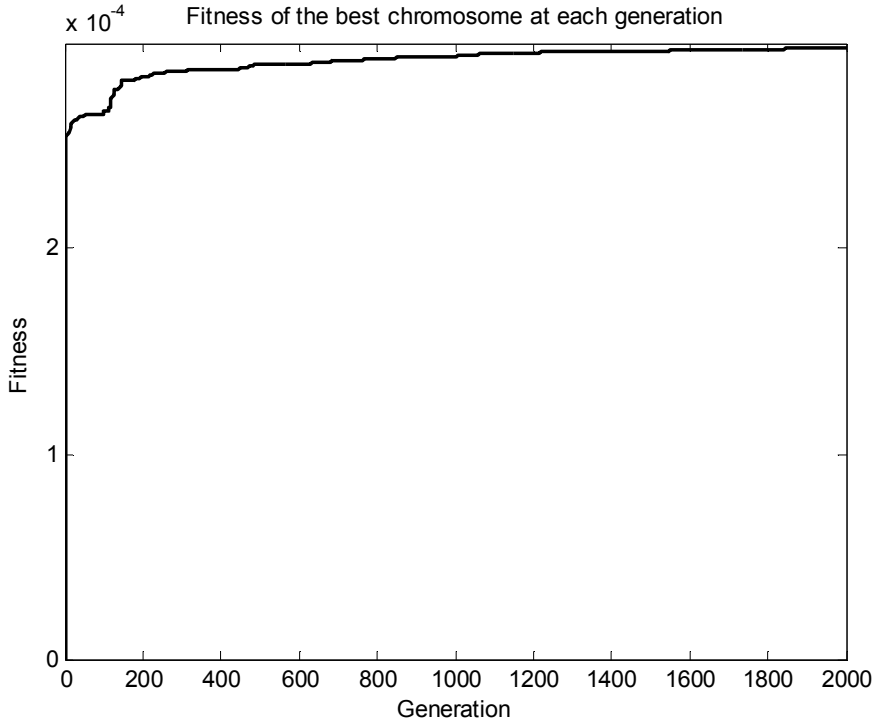

Fig. 3.20: Fitness of the best chromosome at each generation (application 4, the fifth level).

In Fig. 3.21 the performance plots related to the first, the second, the third, the fourth and the fifth levels are shown.

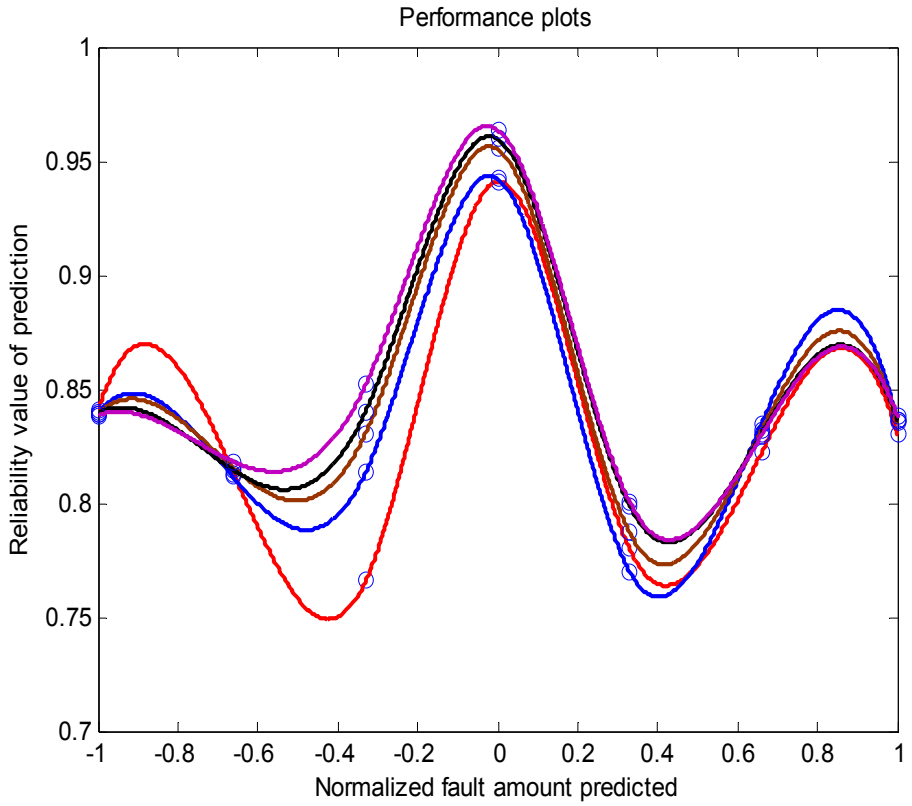

Fig. 3.21: Performance plot for the best agent at each level (application 4): Red for the first level, blue for the second level, brown for the third level, black for the fourth level, violet for the fifth level.

After the agents of each level are created, their efficiency is tested in the validation scenarios. The results are as in the Tables 3.7 and 3.8:

Table 3.7: The results for the fourth application (for the first set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level
Fitness	2.4181×10^{-4}	2.4645×10^{-4}	2.6674×10^{-4}	2.7123×10^{-4}	2.7262×10^{-4}
Cost	4135	4057	3749	3687	3668

Table 3.8: The results for the fourth application (for the second set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level
Fitness	2.6521×10^{-4}	2.7563×10^{-4}	2.9800×10^{-4}	3.0719×10^{-4}	3.1402×10^{-4}
Cost	3770	3628	3355	3255	3184

3.1.5. Application 5:

The main aim in this application is to construct the performance plots in a different way and test its efficiency. Instead of spline interpolation, piecewise linear interpolation is used. The idea is simple: After the success rates are obtained, define small safety regions around their representative points. Draw the performance plot as zero slope lines in these safety regions. Then combine the end points of performance plots at the safety regions corresponding to consecutive success rates via line interpolation to complete the construction. In this application, Equation (3.3) is used to obtain the success rates. The HDM model is carried out for 6 levels. The rule structure at each level is the one used in the first level of the previous applications. In the first level the rule-base is composed of 30 rules while in other levels each rule-base is made up of 20 rules. The GA parameters at each level are the same as the corresponding levels of application 4. The results obtained are shown in Table 3.9.

Table 3.9: The results for the fifth application.

	The first level	The second level	The third level	The fourth level	The fifth level	The sixth level
Fitness (10^{-4})	2.5160	2.7107	2.7972	2.9021	3.0018	3.0424
Cost	3974	3689	3575	3445	3331	3287

Figs. 3.22, 3.23, 3.24, 3.25, and 3.26 show the fitness of the best chromosomes at each generation for the second, the third, the fourth, the fifth and the sixth levels, respectively.

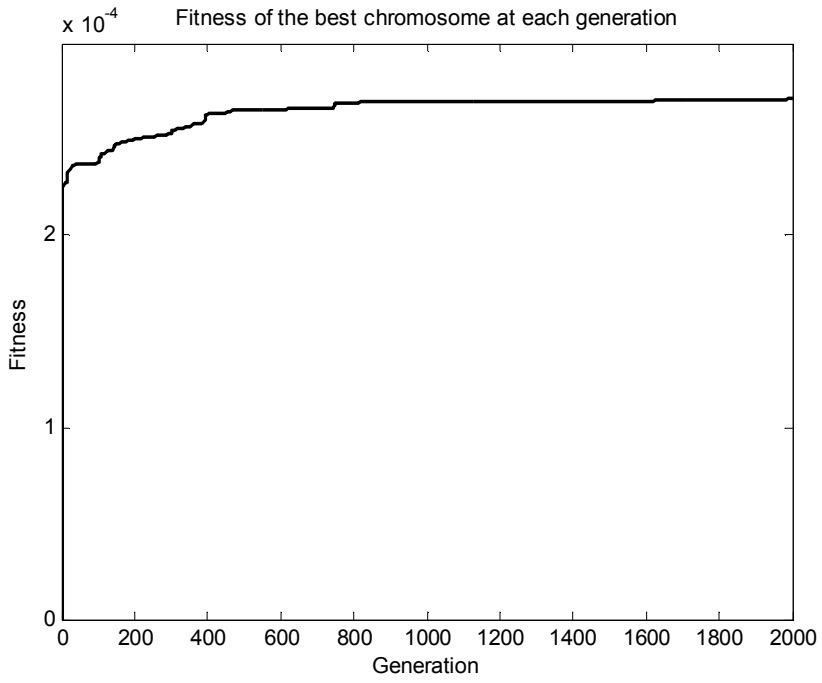

Fig. 3.22: Fitness of the best chromosome at each generation (application 5, the second level).

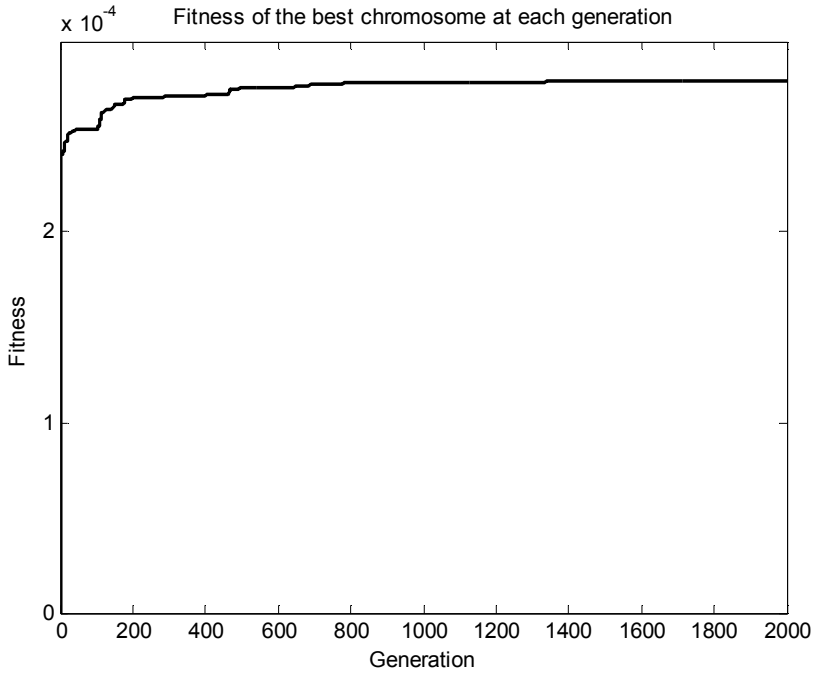

Fig. 3.23: Fitness of the best chromosome at each generation (application 5, the third level).

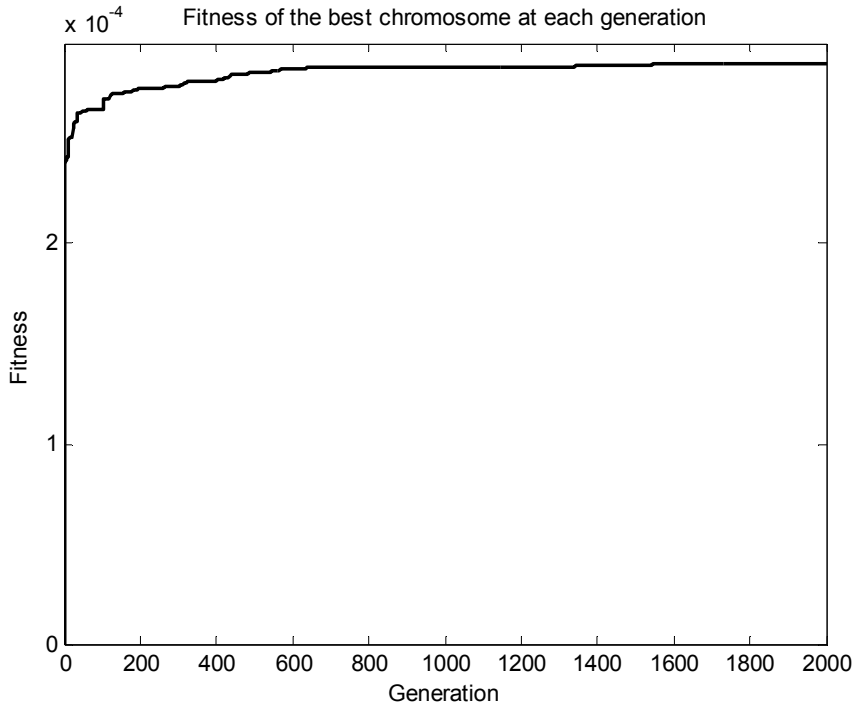

Fig. 3.24: Fitness of the best chromosome at each generation (application 5, the fourth level).

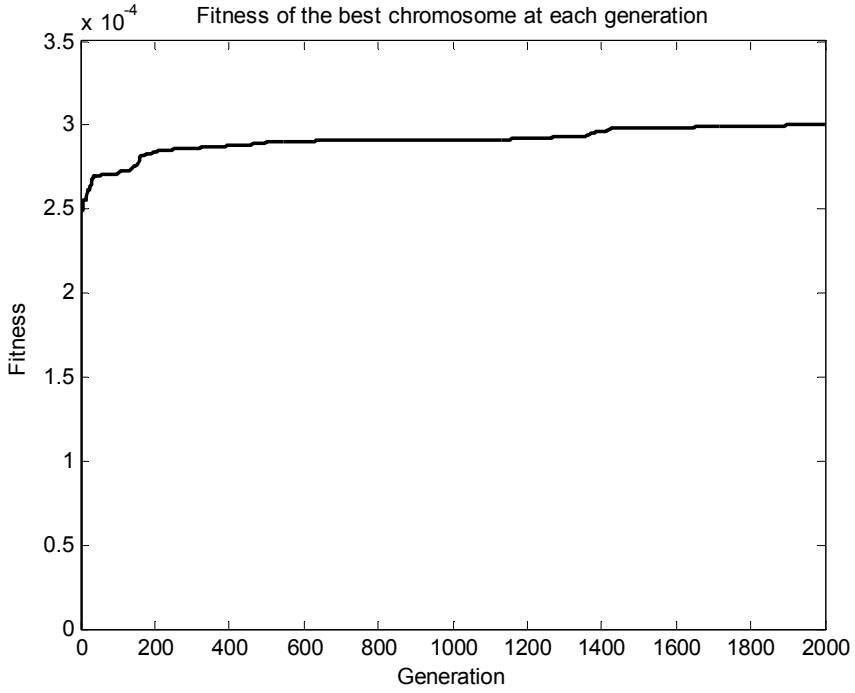

Fig. 3.25: Fitness of the best chromosome at each generation (application 5, the fifth level).

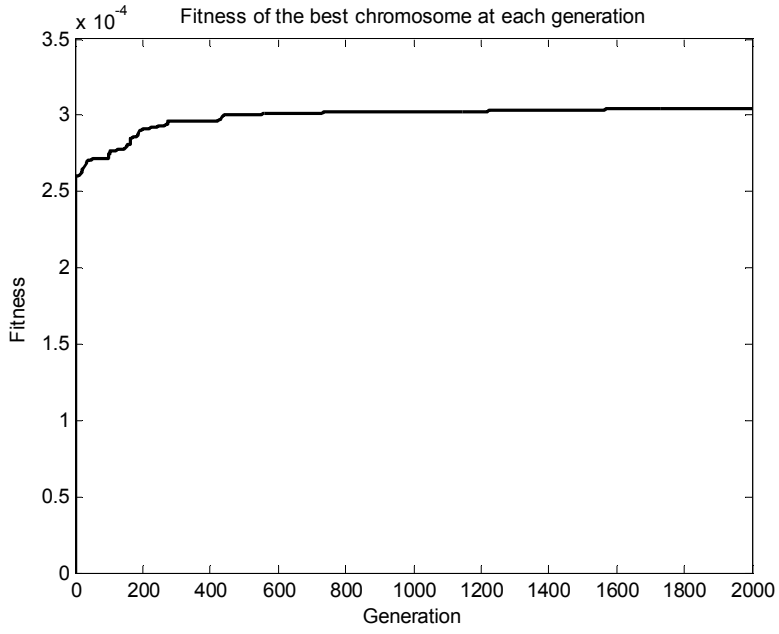

Fig. 3.26: Fitness of the best chromosome at each generation (application 5, the sixth level).

In Fig. 3.27 the performance plots related to the first, the second, the third, the fourth, the fifth and the sixth levels are shown.

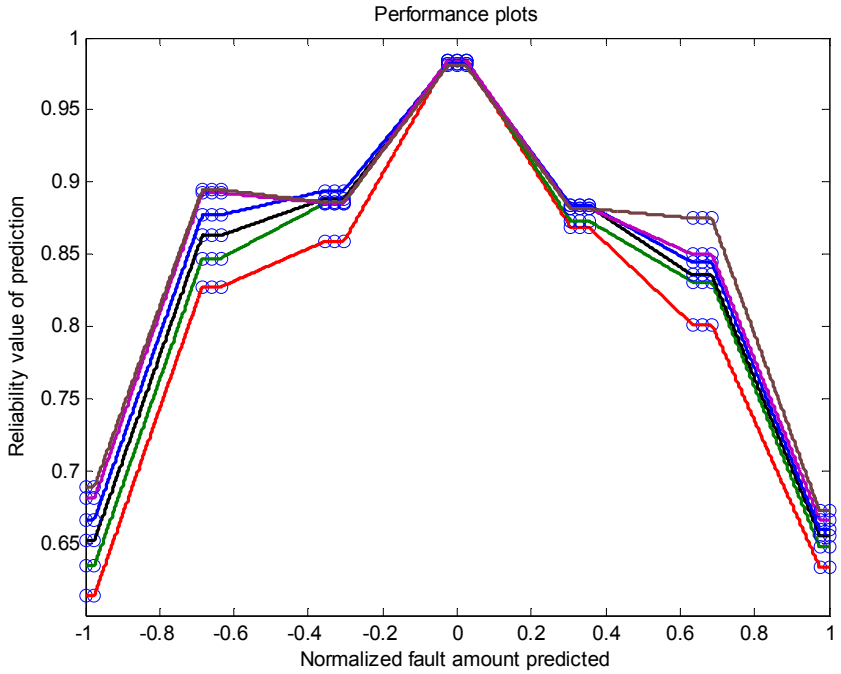

Fig. 3.27: Performance plot for the best agent at each level (application 5): Red for the first level, green for the second level, black for the third level, blue for the fourth level, violet for the fifth level, brown for the sixth level.

After the agents of each level are constructed, their efficiency is tested in the validation scenarios. The results are as in the Tables 3.10 and 3.11:

Table 3.10: The results for the fifth application (for the first set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level	The sixth level
Fitness (10^{-4})	2.4181	2.5250	2.5650	2.6254	2.7028	2.6988
Cost	4135	3960	3898	3808	3699	3705

Table 3.11: The results for the fifth application (for the second set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level	The sixth level
Fitness (10^{-4})	2.6521	2.8509	2.9422	3.0627	3.1659	3.2097
Cost	3770	3507	3398	3265	3158	3115

3.1.6. Application 6

The performance plot for each level agent is created as in application 5. However this time Equation (3.7) is used to obtain the success rates. The HDM model is carried out for 6 levels. The rule structure at each level is the one used in the first level of previous applications. In the first level the rule-base is composed of 30 rules while in other levels each rule-base is made up of 20 rules. The GA parameters at each level are the same as the corresponding levels of application 5. The results obtained are shown in Table 3.12. Figs. 3.28, 3.29, 3.30, 3.31, and 3.32 show the fitness of the best chromosomes at each generation for the second, the third, the fourth, the fifth and the sixth levels, respectively.

Table 3.12: The results for the sixth application.

	The first level	The second level	The third level	The fourth level	The fifth level	The sixth level
Fitness (10^{-4})	2.5160	2.6406	2.8093	2.9654	3.0219	3.0702
Cost	3974	3787	3559	3372	3309	3257

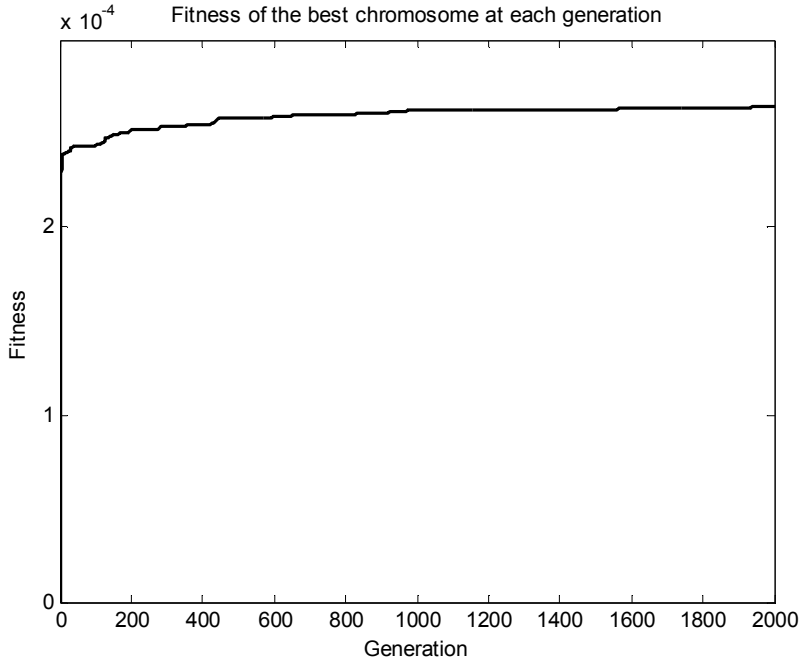

Fig. 3.28: Fitness of the best chromosome at each generation (application 6, the second level).

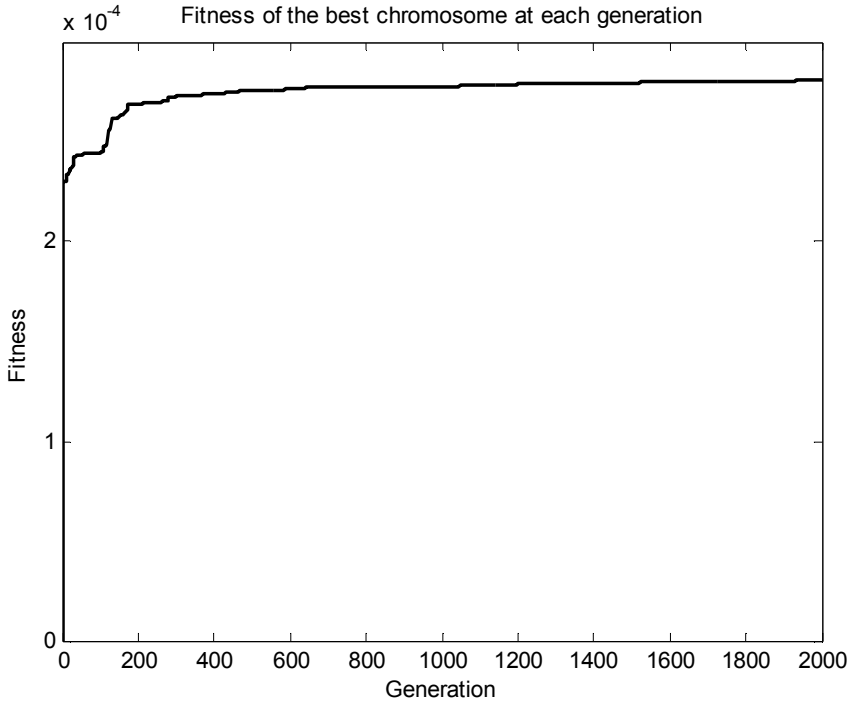

Fig. 3.29: Fitness of the best chromosome at each generation (application 6, the third level).

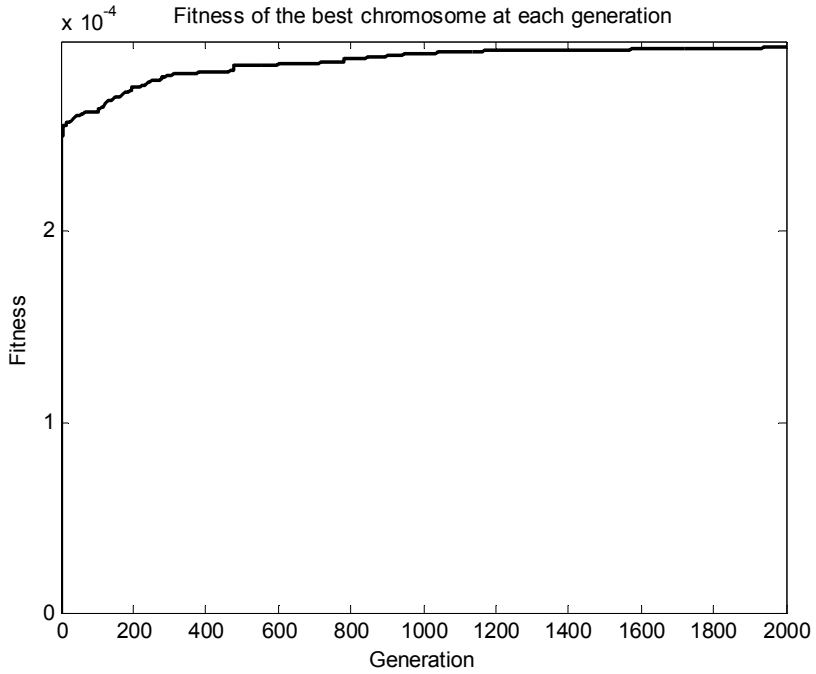

Fig. 3.30: Fitness of the best chromosome at each generation for the fourth level (application 6).

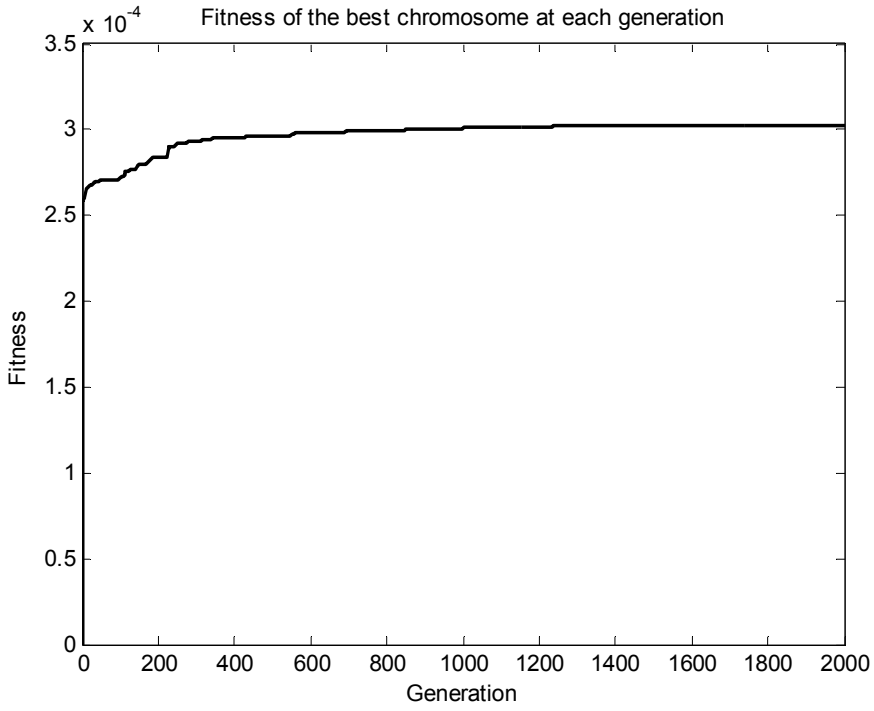

Fig. 3.31: **Fitness of the best chromosome at each generation (application 6, the fifth level).**

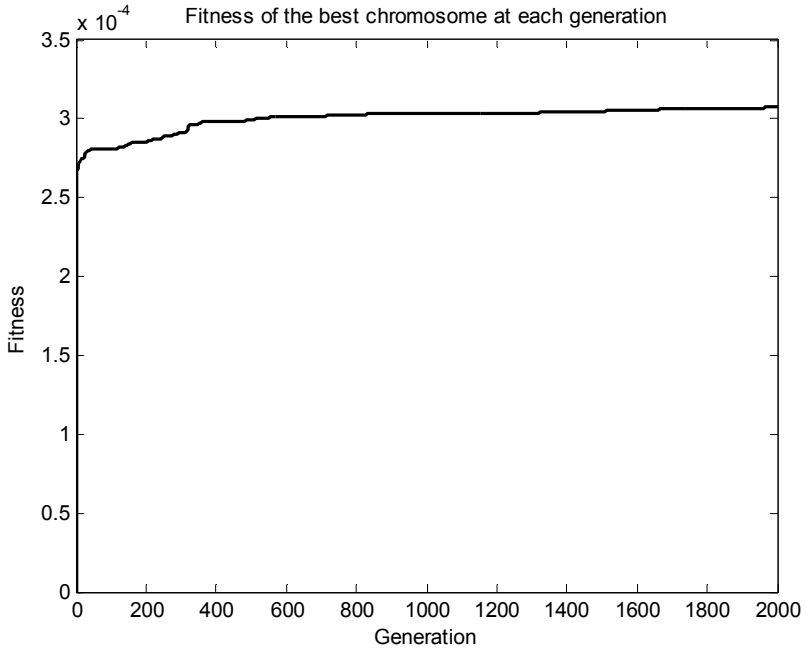

Fig. 3.32: Fitness of the best chromosome at each generation (application 6, the sixth level).

In Fig. 3.33, the performance plots related to the first, the second, the third, the fourth, the fifth, and the sixth levels are shown.

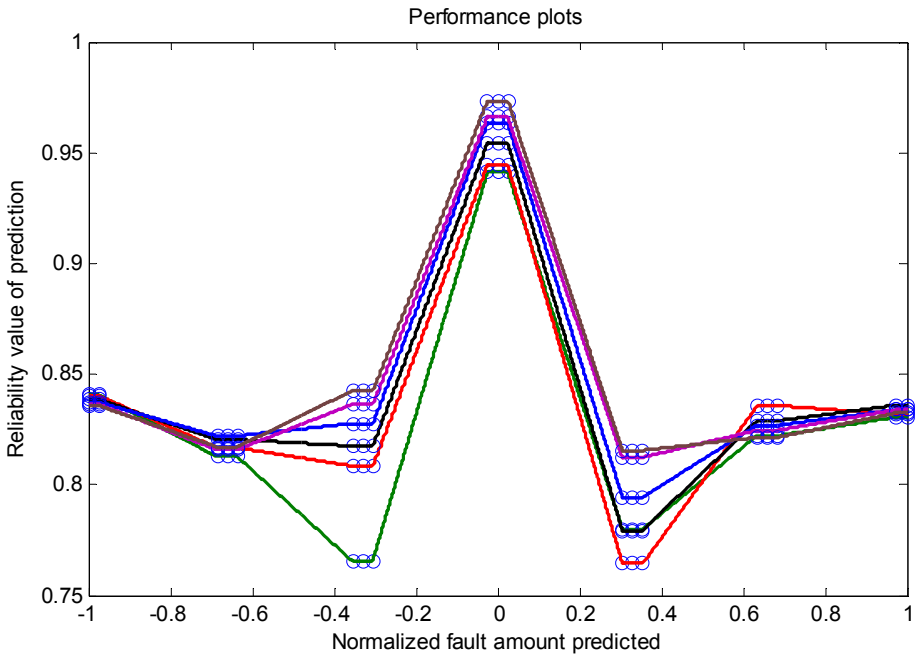

Fig. 3.33: Performance plot for the best agent at each level (application 6): Green for the first level, red for the second level, black for the third level, blue for the fourth level, violet for the fifth level, brown for the sixth level.

After the agents of each level are created, their efficiency is tested in the validation scenarios. The results are as in the Tables 3.13 and 3.14:

Table 3.13: The results for the sixth application (for the first set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level	The sixth level
Fitness (10^{-4})	2.4181	2.4697	2.6216	2.7408	2.7702	2.8312
Cost	4135	4049	3814	3648	3609	3532

Table 3.14: The results for the sixth application (for the second set of validation scenarios).

	The first level	The second level	The third level	The fourth level	The fifth level	The sixth level
Fitness (10^{-4})	2.6521	2.7857	2.9765	3.1247	3.1812	3.2348
Cost	3770	3589	3359	3200	3143	3091

In order to show how the proposed model influence the predicted fault amounts at each level, the predictions of agents at each level for a scenario is plotted in Fig. 3.34.

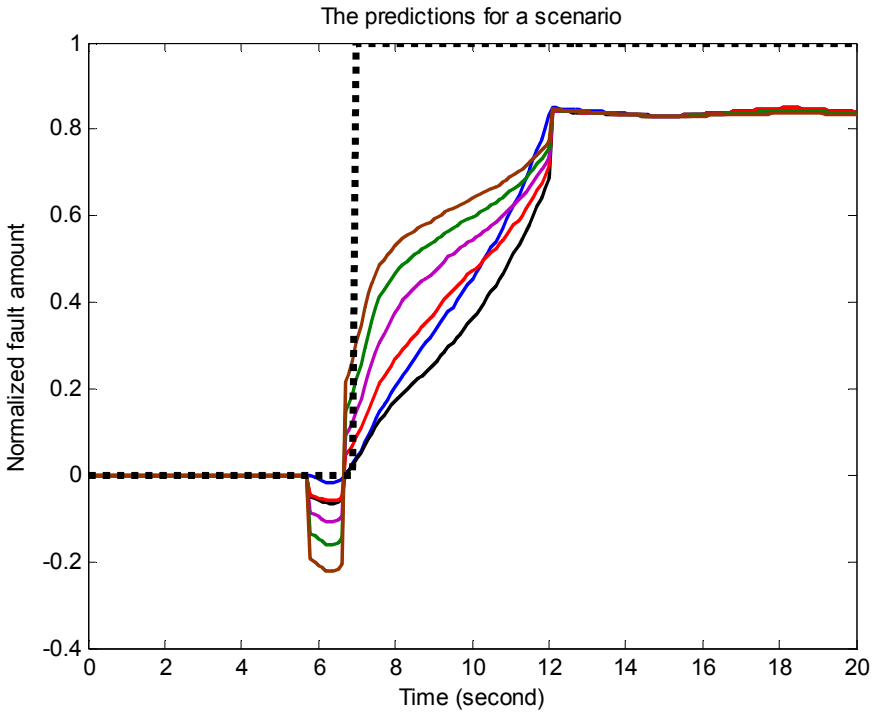

Fig. 3.34: The predictions of each level agent for a scenario. The dotted plot demonstrates the actual normalized fault amount in the first tank whereas the continuous plots correspond to predictions of the agents about the fault amount in the tank at each level (blue for the first level, black for the second level, red for the third level, violet for the fourth level, green for the fifth level, brown for the sixth level).

As seen from Fig. 3.34, the predictions tend to converge to a value between 0.8 and 1 where the real normalized fault amount in the tank is 1. The main reason for this situation is the construction style of the membership functions for positive-big and negative-big attributes, and determination style of output surface using center of area defuzification method. Due to the defuzzification method (i.e., center of area), the center of area of any output surface can at most be 0.844. Membership functions can be taken in different forms to achieve better results.

3.1.7. Application 7

In order to check the influence of the proposed HDM model, a new simulation is performed: In this simulation there are no hierarchical levels. Instead the proposed agent structure is trained using GA optimization technique until 6000 generations (instead of 2000 generations in the first level) and the training is stopped at that point. The structure of the rule-base is assumed to be more complicated since the rule-bases are made up of 70 rules. The rules consist of 4 premise variable and a single consequent variable as the first levels of each application. The agent developed in this application is compared with the agents obtained at the third, the fourth, the fifth and the sixth levels of the last two applications (application 5 and application 6). The simulation parameters and results for application 7 are as follows:

- The crossover and mutation rates are chosen as in application 6.
- Number of rules in each rule-base: 70.
- Number of genes for each chromosome: $5 \times 70 = 350$.
- Number of chromosomes: 40.
- Number of generations: 6000.
- Fitness of the best chromosome throughout the simulation: 2.631×10^{-4}.

Fitness of the best chromosome at each generation is shown in Fig. 3.35.

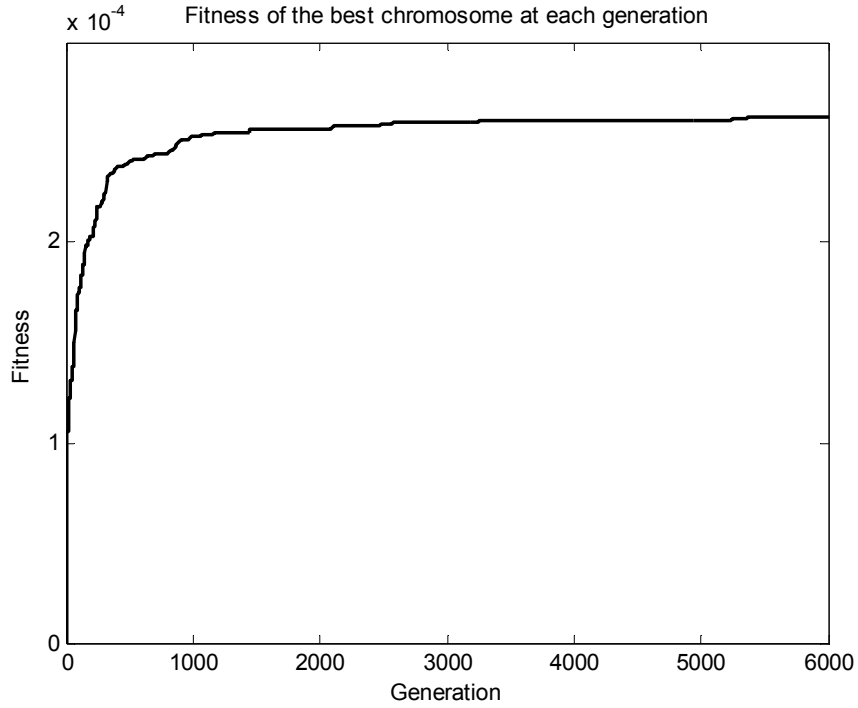

Fig. 3.35: Fitness of the best chromosome at each generation (application 7).

The average fitness of the generation changing with respect to generation is shown in Fig. 3.36.

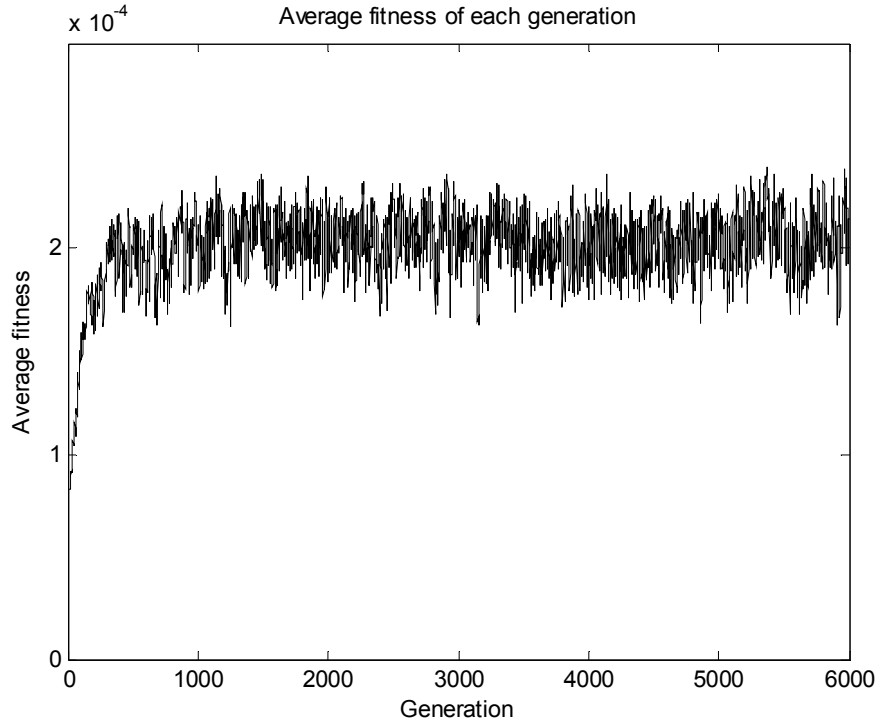

Fig. 3.36: Average fitness of the generation (application 7).

When results of application 7 are compared with results of previous applications, the previous applications which uses the HDM model outperforms application seven which uses a single phase decision making approach that utilized a more sophisticated rule-base structure and trained for larger duration. The computation time for a single generation for this application takes 5-6 times the computation time required to evaluate a generation having 20 rules in the rule-base.

3.2. Noise

In this sub-section, the effect of noise is investigated for the proposed HDM model. Two noisy error data having different distributions are used with three different 'SNR' settings for the training scenarios and the first set of validation scenarios at each level. The details about noisy data can be found in Chapter 2. The agents trained in application 6 are used as the agents.

3.2.1. Uniform Noise

A uniformly distributed random noise is added to the input (the noise error data) with three different SNR choices. Using these noisy error data the performance of the proposed DM model is checked in terms of noise endurance (durability) for the training scenarios. The same procedure is carried out for the first set of validation scenarios. The results are shown in Tables 3.15, 3.16, 3.17, 3.18, 3.19, and 3.20.

Table 3.15: The performance of the first level agent (30 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 1	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.3502×10^{-4}	1.8543×10^{-4}	1.4841×10^{-4}
Fitness (validation)	2.2644×10^{-4}	1.8009×10^{-4}	1.4528×10^{-4}

Table 3.16: The performance of the second level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 2	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.3174×10^{-4}	1.7730×10^{-4}	1.4053×10^{-4}
Fitness (validation)	2.1801×10^{-4}	1.6943×10^{-4}	1.3625×10^{-4}

Table 3.17: The performance of the third level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 3	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.3282×10^{-4}	1.7325×10^{-4}	1.3718×10^{-4}
Fitness (validation)	2.1940×10^{-4}	1.6555×10^{-4}	1.3315×10^{-4}

Table 3.18: The performance of the fourth level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 4	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.2533×10^{-4}	1.6448×10^{-4}	1.3118×10^{-4}
Fitness (validation)	2.1197×10^{-4}	1.5711×10^{-4}	1.2726×10^{-4}

Table 3.19: The performance of the fifth level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 5	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.1602×10^{-4}	1.5847×10^{-4}	1.2804×10^{-4}
Fitness (validation)	2.0323×10^{-4}	1.5140×10^{-4}	1.2425×10^{-4}

Table 3.20: The performance of the sixth level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 6	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.1149×10^{-4}	1.5635×10^{-4}	1.2738×10^{-4}
Fitness (validation)	1.9894×10^{-4}	1.4925×10^{-4}	1.2355×10^{-4}

3.2.2. Gaussian Noise

Gaussian random noise is added to the input (the noise error data) with three different SNR choices. Using these noisy error data the performance of the proposed DM models are checked in terms of noise endurance (durability) for the training scenarios. The same procedure is carried out for the first set of validation scenarios. The results are shown in Tables 3.21, 3.22, 3.23, 3.24, 3.25, and 3.26

Table 3.21: The performance of the first level agent (30 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 1	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.3591×10^{-4}	1.9140×10^{-4}	1.5232×10^{-4}
Fitness (validation)	2.2741×10^{-4}	1.8564×10^{-4}	1.4898×10^{-4}

Table 3.22: The performance of the second level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 2	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.3215×10^{-4}	1.8338×10^{-4}	1.5232×10^{-4}
Fitness (validation)	2.1898×10^{-4}	1.7544×10^{-4}	1.4898×10^{-4}

Table 3.23: The performance of the third level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 3	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.3325×10^{-4}	1.7932×10^{-4}	1.5232×10^{-4}
Fitness (validation)	2.2017×10^{-4}	1.7178×10^{-4}	1.4898×10^{-4}

Table 3.24: The performance of the fourth level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 4	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.2540×10^{-4}	1.7036×10^{-4}	1.5232×10^{-4}
Fitness (validation)	2.1241×10^{-4}	1.6310×10^{-4}	1.4898×10^{-4}

Table 3.25: The performance of the fifth level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 5	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.1576×10^{-4}	1.6356×10^{-4}	1.3070×10^{-4}
Fitness (validation)	2.0328×10^{-4}	1.5685×10^{-4}	1.2698×10^{-4}

Table 3.26: The performance of the sixth level agent (additional 20 rules) for the noise error data (both for the training scenarios and the first set of validation scenarios).

Level 6	SNR (high)	SNR (medium)	SNR (low)
Fitness (training)	2.1077×10^{-4}	1.6048×10^{-4}	1.2935×10^{-4}
Fitness (validation)	1.9875×10^{-4}	1.5393×10^{-4}	1.2571×10^{-4}

As seen from the results, the model proposed becomes ineffective when noisy data is used. This conclusion is due to the fact that the rule-base parts of the agents are used for the noisy error data evaluation at each level to perform the predictions. Besides, the predictions of the rule-base of the current level and the predictions of previous level are combined through the use of performance plots. So the errors in the predictions of each level accumulate.

3.3. Remarks

As observed from the applications, the proposed HDM model demonstrates satisfactory results for the training and the validation scenarios. At each level usually there is an increase in success. On the contrary, the results obtained on the noisy data are unsatisfactory. Error in the decisions seems to accumulate at hierarchical levels and consequently, while passing from a lower level to an upper level performance tend to decrease.

CHAPTER 4

HIERARCHICAL DECISION-MAKING MODEL (RULE-BASE STRUCTURE 2)

In this chapter, the HDM model introduced in Chapter 3 is utilized with a different rule-base structure. In addition, the processing of the data by the rule-bases is also different. In Chapter 3, the raw error data was directly fed into the rule-base and FL is used as the interpolation technique. In this chapter, a rule-base is composed of sequences of cluster indices. It processes the raw data and performs the decisions using a distance measure on the processed data. This new rule-base structure is supposed to improve the performance of the model on noisy data. However, due to additional processing, it is also assumed that the optimization time will be longer compared to the applications in Chapter 3.

4.1. Clustering

The fault detection procedure in this chapter is based on clustering. Essentially, the error data (of the training scenarios) obtained in Chapter 2 is clustered using k-means algorithm. In order to determine the critical number of clusters to be used, the k-means algorithm is applied for different number of cluster choices. The distances of in-cluster samples (any samples of the error data that is closest to a specific cluster center) to their corresponding cluster centers are summed. Then the summations obtained for each cluster are added to obtain a cumulative distance measure for each different choice of the number of clusters. As the number of clusters increases, the cumulative distance measure decreases. After a critical range of values has been attained (when the number of clusters reaches or exceeds a certain value), even if the number of clusters are increased more, the decrease in the cumulative distance measure is negligible. This cumulative distance measure computation is performed for 20 different numbers of cluster choices starting from

20 clusters and finishing at 400 clusters, the incremental changes in the number of clusters being 20. The variation of the cumulative distance measure for different numbers of clusters is shown in Fig. 4.1. We can observe that 400 clusters is a good choice, since the cumulative distance measure plot is almost horizontal there.

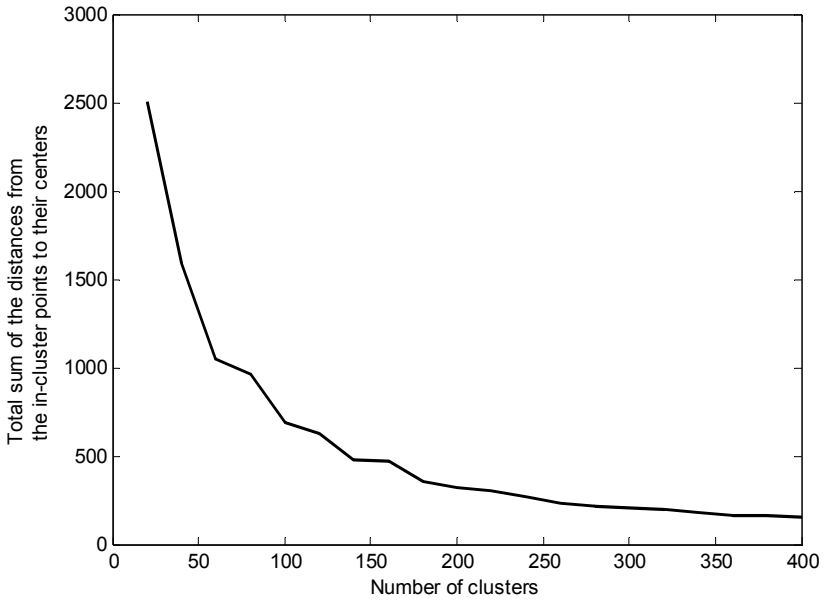

Fig. 4.1: Cumulative sum of total distances of each in-cluster point from the cluster centers.

4.2. The Fault-Detection Procedure:

The rules in the proposed agent structure are made up of sequences. These sequences are composed of two different types of variables. The first type is the cluster index values: They are present in all the terms in the sequences except the last term. The second type is the output attribute: It constitutes the last term of a sequence. An 'output attribute' points out the partial conclusion of the sequence. The combination of a number of sequences forms a rule-base. The rule-base is the dynamic part of the agent structure. The structure of a sequence is shown in Table. 4.1. The combination of sequences (simply a rule-base) is shown in Table. 4.2.

Table 4.1: Basic structure of a sequence: Except for the last term, all the other terms of the sequence indicate the cluster index values activated at corresponding time instants (i.e., $c_{t,x}$, $c_{t-1,x}$….., $c_{t-m+1,x}$). The last term of the sequence is the output attribute of the sequence (i.e., A_x). This sequence has 'm' cluster index values corresponding to time instants starting from 't' ending at 't-m+1' and a single output attribute which is 'A_x').

Time	Sequence_x
t	$c_{t,x}$
t-1	$c_{t-1,x}$
….	
….	
t-m+1	$c_{t-m+1,x}$
Attribute	A_x

Table 4.2: Basic structure of the chromosome.

Time	Sequence_1	Sequence_2	….	….	Sequence_n
t	$c_{t,1}$	$c_{t,2}$	…	…	$c_{t,n}$
t-1	$c_{t-1,1}$	$c_{t-1,2}$	…	…	$c_{t-1,n}$
….			…	…	
….			…	…	
t-m+1	$c_{t-m+1,1}$	$c_{t-m+1,2}$	…	…	$c_{t-m+1,n}$
Attribute	A_1	A_2	…	…	A_n

In Table 4.1, '$c_{i,x}$' is the activated cluster at the corresponding instant 'i' for the sequence. 'A_x' is the output attribute of the sequence. This sequence has a depth of 'm', which is the length of the sequence without the output attribute. Similarly in Table 4.2, each '$c_{i,k}$' value is the activated cluster corresponding to instant 'i' for the k^{th} sequence and 'A_k' is the output attribute of the sequence 'k'.

The depth of a sequence determines the extent it can go back in time. For instance, a sequence of depth 2 can apply to two consecutive samples (i.e., samples at instants 't' and 't-1') of the error data whereas a sequence of depth 10 can apply to

ten consecutive samples (i.e., samples starting from instant 't' ending at instant 't-9') of the error data to calculate a membership value for the sequence at the instant 't'. Hence, each sequence having different depths employs consecutive samples of the error data with different sizes (windows). Therefore, both the sample of present instant and the sample of past instants are together employed to decide on the membership value of a sequence for the present instant. For a sequence of depth '1' only the instantaneous sample (i.e., at instant 't') of the error data is used to determine the membership value of the sequence at instant 't'.

There are three steps to perform the fault prediction using the rule-bases. The first step is calculating the **mutual membership value between a cluster and a sample of the error data**. The second step is assigning **membership values for the sequences** by evaluating a sequential weighted-summation procedure between the cluster index content in the sequence and the window benefiting from the mutual membership values obtained at the first step. The last step is **the aggregation method**. After the membership values for all the sequences in the rule-base have been determined, the aggregation method is utilized to combine the membership values of sequences via the use of corresponding output attributes of the sequences. Eventually, the result obtained at the end of the third step (the aggregation method) is the normalized fault prediction at the first hierarchical level.

Calculation of the **mutual membership value between a cluster and a sample of the error data** is performed as follows:

Step 1- Calculate the distance of each sample of the error data to each cluster center by:

$$d_{data(t),c_i} = \left\| data(t) - c_i \right\|, \tag{4.1}$$

where 'c_i' represents the cluster center, 'data(t)' represents the sample of the error data for the corresponding instant 't', and '$d_{data(t),c_i}$' represents the Euclidean

distance of associated cluster center ‘i’ to the sample of the error data at the instant ‘t’.

Step 2- Calculate the **mutual unnormalized membership value** between a cluster center and a sample of the error data by taking the reciprocal of the distance calculated in Equation (4.1) with

$$unnorm\_mem_{data(t),c_i} = \frac{1}{d_{data(t),c_i}}. \tag{4.2}$$

Step 3- Normalize the membership values using:

$$mem_{data(t),c_i} = \frac{unnorm\_mem_{data(t),c_i}}{\sum_{i=1}^{400} unnorm\_mem_{data(t),c_i}}. \tag{4.3}$$

Step 4- Set very small membership values to 0 (This procedure depends on a selected threshold parameter as shown below):

$$If\ \ mem_{data(t),c_i} \le threshold\ \ then\ \ mem_{data(t),c_i} = 0.$$

The ‘threshold’ is determined as $\frac{1}{400}$ since the number of cluster centers is 400 (sum of membership values of all the clusters for the sample ‘data(t)’ is equal to 1). Then, membership values smaller than or equal to the threshold value are set to 0.

Step 5- Again by re-normalization (similar to the one performed at Step 3) **mutual membership values between a cluster and a sample of the error data** are calculated (this extra normalization application is necessary since at Step 3 some membership values are truncated to 0. In order to add up all the membership values to unity, the strength of the non-zero terms must be increased in accordance

with their influence so that the summation of the of membership values yields 1 again).

Calculation of the membership values for the sequences is related to the assigned **mutual membership values between a cluster and a sample of the error data**. This procedure is described as:

Step 1- Determine the window to be used: If ‘m’ is the depth of the sequence, the window (collection of consecutive samples of the error data) should start from the instant ‘t’ and finish at the instant ‘t-m+1’ (the size of the window should be ‘m×4’; ‘m’ is the depth of the window and ‘4’ is the normalized error values corresponding to each tank at the instants in the window. See Table 4.3 for configuration of a window).

Table 4.3: A window of the error data starting from time instant ‘t’ and lasting at time instant ‘t-m+1’. Each ‘$data_i$’ vector has 4 indices representing the error data values at related tanks for the corresponding time instants.

Window	Data
t	$data_t$
t-1	$data_{t-1}$
….	…….
….	…….
t-m+1	$data_{t-m+1}$

Step 2- Multiply the associated mutual membership values corresponding to the same time instants of the window and the sequence by an exponentially decaying weight function. Perform this multiplication starting from the instant ‘t’ (present time instant) down to the instant ‘t-m+1’. Finally, sum up the multiplicands to obtain **the membership value of the sequence** for the instant ‘t’. In this formulation, time instants closer to present are thought to be more worthy than the earlier time instants. Hence, to diminish the effect of past instants in membership

value assignment, a decaying exponential function is employed, whose impact decreases gradually as time instants go away from the present. In a sequence, clusters situated close to present time instant are more important than the clusters situated far apart from present time. Briefly, in mathematical terms, the ***membership value of the sequence*** for the instant 't' is calculated as:

$$mem\_seq\_x_t = \sum_{k=0}^{m-1} mem_{Window(t-k),Sequence\_x(t-k)} \times e^{-0.1k}. \qquad (4.4)$$

In Equation (4.4), 't' represents the current time and '$mem_{Window(t-k),Sequence\_x(t-k)}$' represents the mutual membership value of cluster center in the sequence activated at the instant 't-k', which is calculated for the sample in the window situated at instant 't-k'. Finally, 'mem_seq_x_t' represents the membership value for the sequence for the instant 't'.

To give an example, applying this procedure to the window at Table 4.3 for the sequence in Table 4.1, the membership value of the 'sequence_x' for the instant 't' is calculated as:

$$mem\_seq\_x_t = mem_{data_t,c_{t,x}} \times e^0 + mem_{data_{t-1},c_{t-1,x}} \times e^{-0.1} + \\ ..... + mem_{data_{t-m+1},c_{t-m+1,x}} \times e^{-(m-1)}. \qquad (4.5)$$

The decision for normalized fault amount at the instant 't' is performed by using the output attributes of the sequences and the normalized membership values assigned for the sequences. This procedure is called **the aggregation of the membership values of the sequences**. Two different aggregation choices are employed in the applications of this chapter. The only difference between these choices is the degree of resolution of the attributes. In one of the choices, the degree of resolution for the attributes is 7 (i.e., 7 membership functions are used for the attributes) whereas for the other choice the degree of resolution is 9 (i.e., 9 membership functions are used for the attributes). The procedure is as follows:

Step 1- For each sequence in the rule-base, calculate the normalized membership values using:

$$norm\_mem\_seq\_x_t = \frac{mem\_seq\_x_t}{\sum_{i=1}^{n} mem\_seq\_i_t}. \tag{4.6}$$

Step 2- If two or more sequences possess the same output attributes, the maximum of the normalized membership values of the corresponding sequences is assigned as the membership degree of the corresponding attribute. For an output attribute possessed by only a single sequence, the normalized membership value of the sequence is assigned as the membership degree of the attribute. For an attribute which is not present in any sequence, 0 is assigned as the membership degree.

Step 3- Determine the output surface using membership functions and the membership degrees assigned to each attribute (similar to the same process in Chapter 3).

Step 4- Calculate the center of area of the output surface which is actually the decision for the normalized fault amount for instant 't'.

The evaluation of the ***center of area*** for the output surface is time-consuming (long integrations have to be performed). Hence, the ***height method*** is applied in order to find an approximate outcome (the output of approximation is declared to be the normalized fault amount decision) for computational reasons. The height method can be applied in various ways. One of the most practical ways depends on multiplying each degree of membership associated with different attributes (having different membership function distributions) by some representative value characterizing some partial information about the attribute. For the sake of simplicity, while utilizing the height method, the argument for the maximum values of each different membership function is used as the representative values of each attribute. The membership functions for each attribute are chosen as in Fig. 3.1 (where the resolution of the attributes is 7). The representative values of attributes are as follows: For negative-big (neg-big) it is -1, for negative-medium

(neg-med) it is -0.6, for negative-small (neg-sml) it is -0.3, for zero it is 0, for positive-small (pos-sml) it is 0.3, for positive-medium (pos-med) it is 0.6 and for positive-big (pos-big) it is 1 (The representative values when the attribute resolution degree is chosen as 9 are as follows: For negative-very-big (NVB) it is -1, for negative-big (NB) it is -0.75, for negative-medium (NM) it is -0.5, for negative-small (NS) it is -0.25, for zero (Z) it is 0, for positive-small (PS) it is 0.25, for positive-medium (PM) it is 0.5 and for positive-big (PB) it is 0.75 and for positive-very-big (PVB) it is 1. The membership functions when the number of membership functions is 9 is shown in Fig. 4.2).

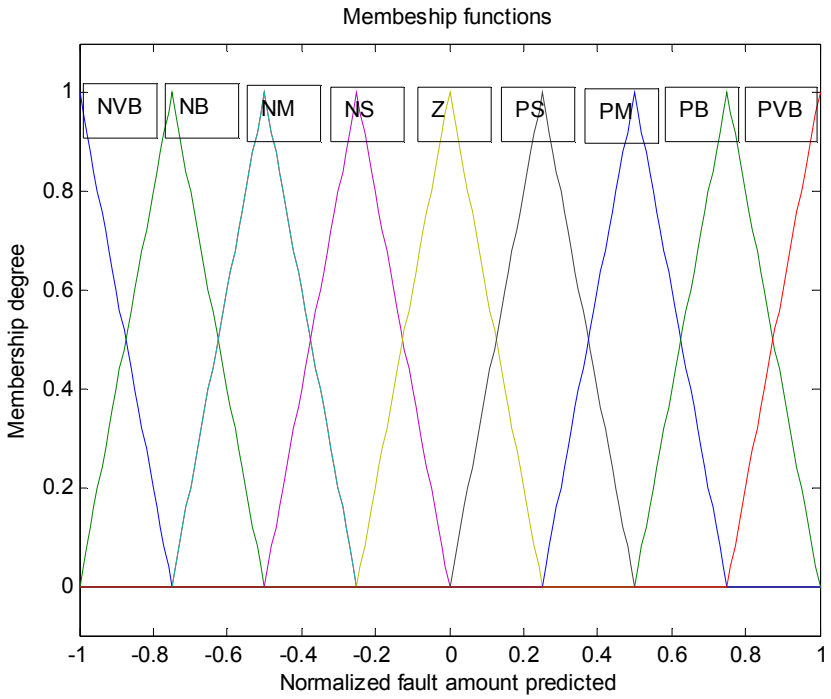

Fig. 4.2: The membership functions for different attributes when the resolution (number of attributes) is 9.

After the representative values for the attributes are determined, the decisions for the fault amounts are calculated by the height method with

$$p\_1(t) = \frac{\left(\sum_{i=neg\_big}^{pos\_big} mem\_\deg\_atr_i(t) \times rep\_val\_atr_i \right)}{\left(\sum_{i=neg\_big}^{pos\_big} mem\_\deg\_atr_i(t) \right)}, \qquad (4.7)$$

where 'p_1(t)' represents the first level prediction for the instant 't', 'mem_deg_atr$_i$(t)' represents membership degree of attribute 'i' at the instant 't' and 'rep_val_atr$_i$' stands for the representative value for attribute 'i'. The weighted addition of representative values of attributes with their membership degree values is performed for all the attributes starting from the negative-big up to the positive-big. If the resolution of attributes is 9, in that case the index values for weighted summation starts from 'negative very big' and continues up to 'positive very big'.

4.3. Applications

All the applications accomplished in this chapter are multi-agent applications, meaning that a different number of agents are developed at each level. Besides, each agent developed at a level has different structures. The decision fusion equation at a level makes use of performance plots of each agent developed at the previous level. There are also some applications in order to compare different decision fusion techniques or to check competence of different agent structures (having different number of sequences in its body or having different sequence depths).

4.3.1. Application 1: Effect of Different Depth Sequences Together in an Agent

A four-level multi-agent structure is constructed in the first application. Three different agents are developed in each level. For the first two levels, each agent possesses a different sequence depth as its dynamic parts. However, for the third and the fourth levels, the dynamic parts of the agents are made up of sequences of mixed depths. The decision-fusion equation at the hierarchical levels is a kind of weighted-averaging equation whose weights are determined according to the associated performance plots of the agents.

4.3.1.1. The First Level Agents:

At the first level, each agent is composed of 10 sequences. One of the agents has a sequence depth of 2, one of the agents has a sequence depth of 5 and one of the agents has a sequence depth of 10. The cost function for the first level simulations is

$$Cost = \sum_{k=1}^{170} \sum_{t=10}^{201} \left| p\_1_k(t) - n\_r\_f_k(t) \right|. \tag{4.8}$$

The time index for the cost function in Equation (4.8) starts from '10' (t = 10) since for each scenario the agent having the longest sequence depth should take information from a window of depth '10'. Hence the cost function is calculated ignoring the first 10 time instants of each scenario. In Equation (4.8), '$p\_1_k(t)$' is the normalized first level fault prediction for instant 't' at k^{th} scenario and '$n\_r\_f_k(t)$' is the normalized real fault for time instant 't' at k^{th} scenario. GA is used in order to optimize the agent structures as in Chapter 3. Each rule-base in a population of GA simulations is represented by a chromosome. The sequences of a rule-base are concatenated to each other one by one, starting with the first sequence until the last sequence and eventually the chromosome structure is obtained. What are updated in the optimization process are the variables in the

rule-base (the cluster index values or the output attribute values of the sequences). Fitness of each chromosome is calculated by taking the reciprocal of the cost value calculated for the associated rule-base.

In the GA simulations, the population sizes are chosen to be 40. The mutation rate for each gene is chosen to be 2 %, the crossover and reproduction rates are selected as 90 % and 10 %, respectively. Roulette wheel selection is performed and the elitist method is used for the best chromosome at each generation. The simulations are run for 2000 generations. The chromosomes having the best fitness values at the end of simulations are declared as the agents of the first level. The fitness and cost values of agents in the simulations are shown in Table 4.4.

Table 4.4: The cost and fitness values of the first level agents.

	10 sequence of depth 2	10 sequence of depth 5	10 sequence of depth 10
Fitness	1.769×10^{-4}	1.886×10^{-4}	1.874×10^{-4}
Total cost	5652.9	5302,2	5336,1

4.3.1.2. The Second Level Agents

The decisions of the first level agents are used in the development of new second level agents. A decision fusion technique is applied to obtain 3 different second level agents in 3 new simulations. The developed agents have dynamic and static parts. The dynamic part of an agent is enclosed in a rule-base similar to the rule-bases at the first level (a second level rule-base having 10 sequences of depth 2, a second level rule-base of 10 sequences of depth 5, and a second level rule-base of 10 sequences of depth 10). These rule-bases are subject to changes due to optimization. The static part of an agent corresponds to the first level decisions and the reliability values of the first level decisions attained using the performance

plots for the first level decisions. In order to fuse the first level decisions with the decisions of a developing rule-base, performance plots are used. The construction of the performance plots is carried out in the same manner as in the first application of Chapter 3. The performance plot for the first level agent of depth 2 is shown at Fig. 4.3.

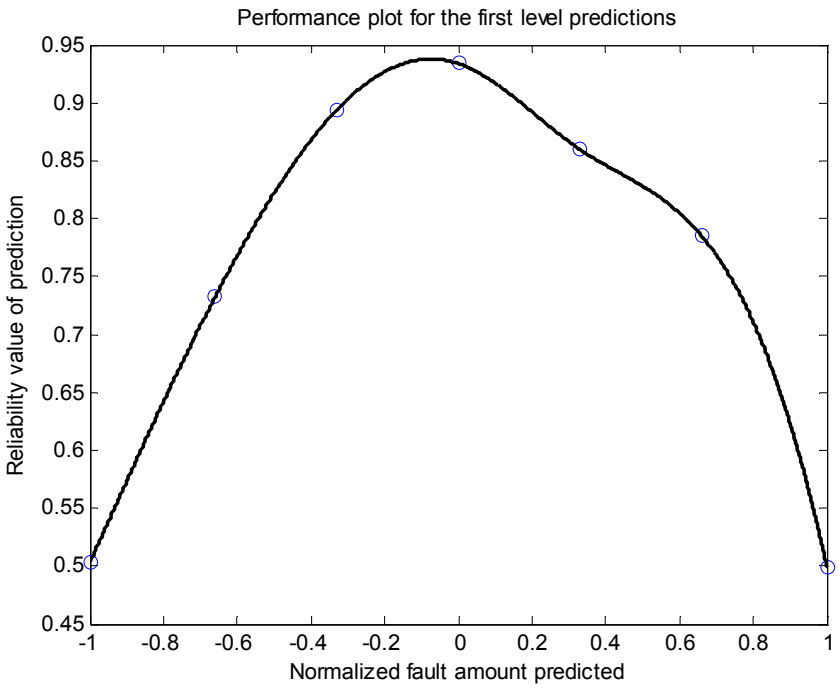

Fig. 4.3: Performance plot for the first level agent having a rule-base of sequence depth of 2.

The determination of the decision of a second level **potential** agent at time instant 't' is carried out using the following procedure:

- Determine the reliability values of the decisions of three first level agents from their performance plots.
- Determine the decisions of the rule-base being trained in this level.
- Fuse the decisions of previous level agents and the decisions of being trained rule-base in this level.

The fusion function for determining the second level prediction for instant 't' is as:

$$p\_2(t) = p_{r\_b}(t) \times (1 - \frac{r\_1_{2r}(t) + r\_1_{5r}(t) + r\_1_{10r}(t)}{3}) + \\ (p\_1_{2r}(t) \times r\_1_{2r}(t) + p\_1_{5r}(t) \times d\_1_{5r}(t) + p\_1_{10r}(t) \times d\_1_{10r}(t)) \times \frac{1}{3}, \quad (4.9)$$

where, 'p_2(t)' represents the second level prediction for the time instant, '$p_{r\_b}(t)$' represents the prediction of the new developing rule-base, '$p\_1_{2r}(t)$', '$p\_1_{5r}(t)$' and '$p\_1_{10r}(t)$' represent the first level predictions of the first level agents having different sequence depths and 'd_1_{2r}', 'd_1_{5r}' and 'd_1_{10r}' represent the corresponding reliability values of the first level predictions obtained from the performance plots of the first level agents. As can be understood from the fusion equation, the second level decision is a weighted convex summation. If the reliability values of the first level predictions are small, more chance is given to the newly developing rule-base to update the second level prediction.

GA is used in order to optimize the dynamic part of the agent structures of the second level and the representation of the rule-base as a chromosome is similar to the first level. The cost function for a chromosome is similar to (8). However, instead of the first level predictions, the second level predictions are embedded in the cost function as:

$$Cost = \sum_{k=1}^{170} \sum_{t=10}^{201} |p\_2_k(t) - n\_r\_f_k(t)|, \quad (4.10)$$

where '$p\_2_k(t)$' is the normalized second level fault prediction for time instant 't' at k^{th} scenario and '$n\_r\_f_k(t)$' is the normalized real fault for time instant 't' at k^{th} scenario. It should be noted that the cost function contains the predictions obtained as the result of fusion equation not the prediction of the new developing rule-base alone. The fitness of each chromosome is calculated by taking the reciprocal of the cost value calculated for the associated chromosome. The GA parameters are taken

the same as in the first level. The success indices of best chromosomes obtained at the end of simulations are shown in Table 4.5.

Table 4.5: The cost and fitness values of the second level agents.

	10 sequence of depth 2	10 sequence of depth 5	10 sequence of depth 10
Fitness	2.2019×10^{-4}	2.1577×10^{-4}	2.062×10^{-4}
Total cost	4541.5	4634.5	4849.8

4.3.1.3. The Third Level Agents

It is possible to use variable depth sequences in a single rule-base. In this level, the predictions of the agents obtained at the second level are used to help generation of 3 different third level agents. However, the rule-bases proposed in the simulations are composed of composite sequences of different depths. The three new rule-bases developed in the third level are as follows: The first rule-base (a rule-base of 12 sequence: 6 sequences of depth 2, 3 sequences of depth 5, 3 sequences of depth 10), the second rule-base (a rule-base of 12 sequence: 3 sequences of depth 2, 6 sequences of depth 5, 3 sequences of depth 10), the third rule-base (a rule-base of 12 sequences: 3 sequences of depth 2, 3 sequences of depth 5, 6 sequences of depth 10).

The fusion equation is the similar to Equation (4.7): The only difference is that the second level predictions of best agents and their corresponding reliability values are used to find the third level predictions. Since the sequences have variable depths, an extra normalization procedure is necessary for finding the membership values of sequences. So Equation (4.4) is revised and the following equation is obtained:

$$mem\_seq\_x_t = \frac{\sum_{k=0}^{m-1} mem_{Window\ (t-k),\ Sequence\ \_x(t-k)} \times e^{-0.1k}}{\sum_{k=0}^{m-1} e^{-0.1k}}. \quad (4.11)$$

The other details about the simulation parameters are the same as the second level simulations. The performance indices for the agents at the simulations are shown in Table 4.6:

Table 4.6: The cost and fitness values of the third level agents.

	The first agent	The second agent	The third agent
Fitness of best agent	2.322×10^{-4}	2.285×10^{-4}	2.340×10^{-4}
Total cost	4306.6	4376.3	4273.5

4.3.1.4. The Fourth Level Agents

The simulations carried out in this level are similar to the third level simulations (3 new agents are developed whose structures are similar to the agents developed in the third level. Besides, the GA parameters are chosen similar to the third level simulations). The results obtained for the best agents in the simulations are shown in Table 4.7.

Table 4.7: The cost and fitness values of the fourth level agents.

	The first agent	The second agent	The third agent
Fitness	2.428×10^{-4}	2.420×10^{-4}	2.444×10^{-4}
Total cost	4118	4132.3	4091.6

4.3.2. Application 2: A Standard Hierarchical Multi-Agent Approach with Non-Changing Agent Structures in the Level

The first two levels of Application 2 are copied directly from the first and the second levels of Application 1. In the third, the fourth, the fifth and the sixth levels, the rule-bases are formed from the same depth sequences as in the first and the second level of the first application. Therefore the rule-bases are made of 10 sequences of depth 2, depth 5 and depth 10 respectively. The result of the third level is shown in Table 4.8:

Table 4.8: The fitness values of the third level agents.

	The first agent	The second agent	The third agent
Fitness	2.3494×10^{-4}	0.2279×10^{-4}	2.2295×10^{-4}

The result of the fourth level is shown in Table 4.9:

Table 4.9: The fitness values of the fourth level agents.

	The first agent	The second agent	The third agent
Fitness	2.4021×10^{-4}	2.3807×10^{-4}	2.3123×10^{-4}

The result of the fifth level is shown in Table 4.10:

Table 4.10: The fitness values of the fifth level agents.

	The first agent	The second agent	The third agent
Fitness	2.4932×10^{-4}	2.4308×10^{-4}	2.3821×10^{-4}

The result of the sixth level is shown in Table 4.11:

Table 4.11: The fitness values of the sixth level agents.

	The first agent	The second agent	The third agent
Fitness	2.5458×10^{-4}	2.4735×10^{-4}	2.4662×10^{-4}

As seen from the results, agents having rule-bases with smaller sequence depths are more effective (have better fitness values) at each level. There may be two main reasons for that fact. Firstly, it is hard to optimize rule-bases having many number of variables compared to rule-bases having moderate number of variables. Secondly, constructing and setting the relations inside a sequence is easier if the

sequence depth is smaller. For these two reasons rule-bases having smaller sequence depths are more successful.

4.3.3. Application 3: Comparison of Different Sequence Depths at the First Level

In this application, 6 different first level agents, each having different sequence depths are compared with each other. Each rule-base has 10 sequences. The first 3 agents are the ones developed in the first level of Application 1 (the depths are 2, 5, and 10, respectively). The other 3 agents possess depths of 1, 3, and 7. The cost function to develop the agents is given in Equation (4.8). The GA parameters are the same as the previous applications. The results are tabulated in Table 4.12.

Table 4.12: The comparison of the first level agents having different depths.

The first level agents	Agent (depth 1)	Agent (depth 2)	Agent (depth 3)	Agent (depth 5)	Agent (depth 7)	Agent (depth 10)
Fitness (10^{-4})	2.1396	1.7688	2.0184	1.8858	1.8627	1.8738

As seen from the fitness values of the agents, there is generally a decline in fitness values when the agent structure becomes complicated (depth increases while the number of sequences remains the same when the GA simulations are carried out for 2000 generations). It seems that optimization of a chromosome having a higher number of genes necessitates usage of more computation time to attain good results. There is only one exception for this result. The agent having a depth of 2 has a terrible fitness value. A GA search in that simulation may be sticking on some local minimum due to the cost function and it might not find its way out of the locality with the facilities of the GA search for that simulation.

4.3.4. Application 4: Comparison of Different Fusion Techniques

In this section, the effect of changing the fusion equation is tested for a specific level. In order to help development of the second level agents, the first level agents of the first application are chosen. The first level agents' decisions are used in order to obtain two different second level agents. Both of these second level agents possess a dynamic part made up of 10 sequences of depth 2. The fusion equations to develop these agents are different. The fusion technique to develop the first agent is shown in Equation (4.9) whereas the fusion technique used to develop the second agent is obtained by

$$\begin{aligned} p\_2(t) = {} & p_{r\_b}(t) \times (1 - \max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t))) + \\ & \arg(\max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t))) \times \max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t)), \end{aligned} \tag{4.12}$$

where

$$\arg(\max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t))) = p\_1_{2r}(t) \text{ if } \max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t)) = r\_1_{2r}(t),$$

$$\arg(\max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t))) = p\_1_{5r}(t) \text{ if } \max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t)) = r\_1_{5r}(t),$$

$$\arg(\max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t))) = p\_1_{10r}(t) \text{ if } \max(r\_1_{2r}(t), r\_1_{5r}(t), r\_1_{10r}(t)) = r\_1_{10r}(t).$$

The fusion equation (i.e., Equation (4.12)) checks for the maximum reliability among the previous level predictions. The prediction whose reliability value is the highest is chosen as the main (actual) prediction of the first level agents and than the prediction with its associated reliability value is inserted into the fusion equation.

The GA parameters in the simulations are chosen to be the same as the previous applications. The simulation using Equation (4.9) is the replica of the second level of Application 1. The results of this simulation can be monitored on the first column of Table 4.5 (Fitness = 2.2019×10^{-4}, Cost = 4541.5). The fitness for the second agent is agent is 1.9646×10^{-4}. As seen from the results, the weighted averaging fusion equation is more successful then the maximum taking in fusion.

4.3.5. Application 5: Effect of Increasing Number of Sequences and Resolution (Number of Membership Functions for Output Attributes)

In this new application, instead of three agents, two agents are developed at each level. The rule-base of one of the agents is made of sequences of depth 1 whereas the second agent possess a rule-base made of sequences of depth 2 (The main reason to choose rule-bases with small depth values is based on the results obtained in Application 3. Rule-bases with small sequence depths seem to be more successful compared to rule-bases with long sequence depths). The rule-bases are made of 20 sequences. The fusion equation while moving from the first level to the second level is given by

$$p\_2(t) = p_{r\_b}(t)\times(1-\frac{r\_1_{1r}(t)+r\_1_{2r}(t)}{2})+ \\ (p\_1_{1r}(t)\times r\_1_{1r}(t)+p\_1_{2r}(t)\times r\_1_{2r}(t))\times\frac{1}{2}, \tag{4.13}$$

where '$p_{r\text{-}b}$' is the prediction of the new rule-base in the second level, 'p_1_{1r}' is the prediction of the first agent having sequence depth of 1, 'p_1_{2r}' is the prediction of the first level agent having sequence depth of 2, 'r_1_{1r}' is the reliability value of prediction of the first level agent having sequence depth of 1, 'r_1_{2r}' is the reliability value of prediction of the first level agent having sequence depth of 2, and 'p_2' is the prediction of the second level agent. The fusion equation between any consecutive levels is similar to Equation (4.13). In this application, the resolution degree of attributes is determined as 9 (shown in Fig. 4.2). The GA simulations at each level are run for 2500 generations whereas the other GA parameters are the same as in previous applications. However, in this application,

the rule-base with the deepest sequence has a sequence depth of 2. Hence the time index in the cost function (i.e., Equation (4.10)) can start from t = 2. The results for training scenarios are tabulated at Table 4.13.

Table 4.13: Fitness of the agents for the training scenarios.

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	2.0419×10^{-4}	1.9351×10^{-4}
Level 2	2.2382×10^{-4}	2.1766×10^{-4}
Level 3	2.3438×10^{-4}	2.2805×10^{-4}
Level 4	2.4010×10^{-4}	2.3117×10^{-4}
Level 5	2.4332×10^{-4}	2.4031×10^{-4}
Level 6	2.4571×10^{-4}	2.4618×10^{-4}

As seen from the results, increasing the number of variables to be optimized (number of sequences) or the choices for the variables (resolution of attributes) seem to decrease the convergence speed. The fitness values observed for the agents at different levels of Application 2 are generally better than the fitness values of agents at the same levels of Application 5. However it should not be forgotten that the cost functions of Applications 2 and 5 starts from different time index values (number of samples checked in the fifth application is more numerous). So making a consistent comparison for the success of the applications is troublesome. The results for the first set of validation scenarios are shown in Table 4.14.

Table 4.14: Fitness of the agents for the first set of validation scenarios.

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	2.0080×10^{-4}	1.8764×10^{-4}
Level 2	2.1437×10^{-4}	2.1053×10^{-4}
Level 3	2.2503×10^{-4}	2.1774×10^{-4}
Level 4	2.2978×10^{-4}	2.1960×10^{-4}
Level 5	2.3284×10^{-4}	2.2744×10^{-4}
Level 6	2.3337×10^{-4}	2.3190×10^{-4}

The results for the second set of validation scenarios are shown in Table 4.15.

Table 4.15: Fitness of the agents for the second set of validation scenarios.

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	2.1735×10^{-4}	2.0333×10^{-4}
Level 2	2.3790×10^{-4}	2.2962×10^{-4}
Level 3	2.4850×10^{-4}	2.4104×10^{-4}
Level 4	2.5482×10^{-4}	2.4466×10^{-4}
Level 5	2.5771×10^{-4}	2.5532×10^{-4}
Level 6	2.6072×10^{-4}	2.6140×10^{-4}

In Fig. 4.4, the normalized fault prediction for a scenarios is plotted when the developed agents at each hierarchical level having a sequence depth of 1 are used as the decision makers.

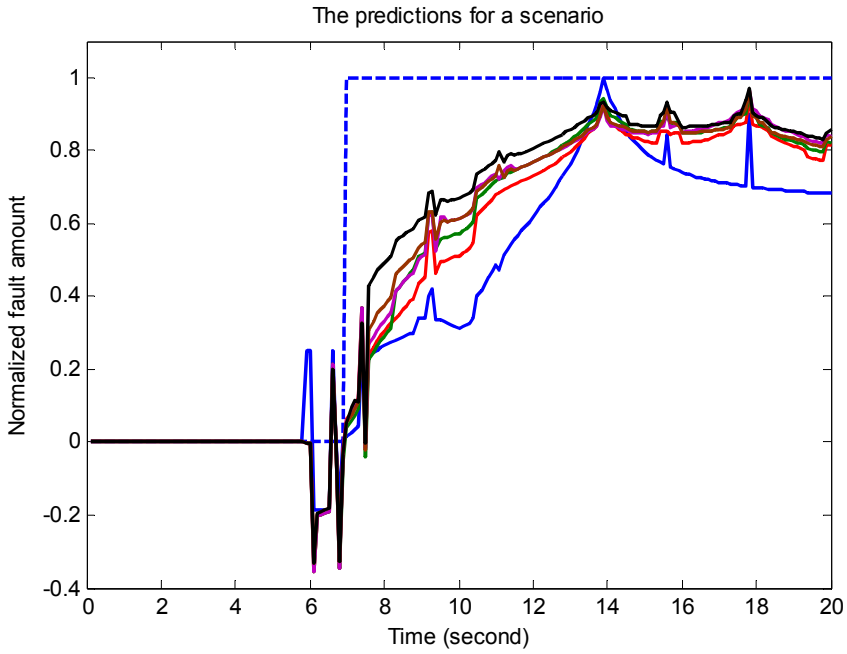

Fig. 4.4: The normalized fault predictions for a scenario when the developed agents at each hierarchical level (having a sequence depth of 1) are used as the decision maker. The dotted blue shows the real fault amount, the blue line shows the first level predictions, the red line shows the second level predictions, the green line shows the third level predictions, the violet line shows the fourth level predictions, the brown line shows the fifth level predictions, and the black line shows the sixth level predictions.

4.4. Noise

The performance of the proposed HDM model for noisy error data is also examined. Uniform noise and Gaussian noise are added to the error data both for the training scenarios and the first set of validation scenarios (formation of noisy error data is explained in Chapter 2). The agents developed at the fifth application are used in order to see the influence of noise in the performance of the decisions. The results for the training scenarios with uniform noise (SNR high) are shown in Table 4.16.

Table 4.16: The fitness values for the training scenarios with uniform noise (SNR high).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	2.0579×10^{-4}	1.9895×10^{-4}
Level 2	2.2469×10^{-4}	2.1978×10^{-4}
Level 3	2.3581×10^{-4}	2.2893×10^{-4}
Level 4	2.4167×10^{-4}	2.3208×10^{-4}
Level 5	2.4473×10^{-4}	2.4088×10^{-4}
Level 6	2.4446×10^{-4}	2.4478×10^{-4}

The results for the first set of validation scenarios with uniform noise (SNR high) are shown in Table 4.17.

Table 4.17: The fitness values for the first set of validation scenarios with uniform noise (SNR high).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	2.0039×10^{-4}	1.9150×10^{-4}
Level 2	2.1376×10^{-4}	2.1100×10^{-4}
Level 3	2.2497×10^{-4}	2.1734×10^{-4}
Level 4	2.2974×10^{-4}	2.1932×10^{-4}
Level 5	2.3273×10^{-4}	2.2640×10^{-4}
Level 6	2.3076×10^{-4}	2.2939×10^{-4}

The results for the training scenarios with uniform noise (SNR medium) are shown in Table 4.18.

Table 4.18: The fitness values for the training scenarios with uniform noise (SNR medium).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.6360×10^{-4}	1.8767×10^{-4}
Level 2	1.8867×10^{-4}	1.8476×10^{-4}
Level 3	1.8917×10^{-4}	1.8967×10^{-4}
Level 4	1.9435×10^{-4}	1.8889×10^{-4}
Level 5	1.8843×10^{-4}	1.8746×10^{-4}
Level 6	1.8698×10^{-4}	1.9056×10^{-4}

The results for the first set of validation scenarios with uniform noise (SNR medium) are shown in Table 4.19.

Table 4.19: The fitness values for the first set of validation scenarios with uniform noise (SNR medium).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.5664×10^{-4}	1.8085×10^{-4}
Level 2	1.7890×10^{-4}	1.7672×10^{-4}
Level 3	1.7950×10^{-4}	1.8017×10^{-4}
Level 4	1.8441×10^{-4}	1.7870×10^{-4}
Level 5	1.7841×10^{-4}	1.7634×10^{-4}
Level 6	1.7619×10^{-4}	1.7921×10^{-4}

The results for the training scenarios with uniform noise (SNR low) are shown in Table 4.20.

Table 4.20: The fitness values for the training scenarios with uniform noise (SNR low).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.3995×10^{-4}	1.5723
Level 2	1.5663×10^{-4}	1.5634×10^{-4}
Level 3	1.4886×10^{-4}	1.5852×10^{-4}
Level 4	1.5464×10^{-4}	1.5418×10^{-4}
Level 5	1.4996×10^{-4}	1.5051×10^{-4}
Level 6	1.4935×10^{-4}	1.5269×10^{-4}

The results for the first set of validation scenarios with uniform noise (SNR low) are shown in Table 4.21.

Table 4.21: The fitness values for the first set of validation scenarios with uniform noise (SNR low).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.3578×10^{-4}	1.5245×10^{-4}
Level 2	1.5066×10^{-4}	1.5105×10^{-4}
Level 3	1.4301×10^{-4}	1.5265×10^{-4}
Level 4	1.4843×10^{-4}	1.4803×10^{-4}
Level 5	1.4381×10^{-4}	1.4386×10^{-4}
Level 6	1.4298×10^{-4}	1.4589×10^{-4}

The results for the training scenarios with Gaussian noise (SNR high) are shown in Table 4.22.

Table 4.22: The fitness values for the training scenarios with Gaussian noise (SNR high).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	2.0302×10^{-4}	1.8793×10^{-4}
Level 2	2.1702×10^{-4}	2.0974×10^{-4}
Level 3	2.2149×10^{-4}	2.1637×10^{-4}
Level 4	2.2663×10^{-4}	2.1700×10^{-4}
Level 5	2.2373×10^{-4}	2.2146×10^{-4}
Level 6	2.2331×10^{-4}	2.2384×10^{-4}

The results for the first set of validation scenarios with Gaussian noise (SNR high) are shown in Table 4.23.

Table 4.23: The fitness values for the first set of validation scenarios with Gaussian noise (SNR high).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.9739×10^{-4}	1.8651×10^{-4}
Level 2	2.1106×10^{-4}	2.0702×10^{-4}
Level 3	2.2118×10^{-4}	2.1502×10^{-4}
Level 4	2.2647×10^{-4}	2.1636×10^{-4}
Level 5	2.2938×10^{-4}	2.2339×10^{-4}
Level 6	2.2779×10^{-4}	2.2714×10^{-4}

The results for the training scenarios with Gaussian noise (SNR medium) are shown in Table 4.24.

Table 4.24: The fitness values for the training scenarios with Gaussian noise (SNR medium).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.6990×10^{-4}	1.8793×10^{-4}
Level 2	1.9335×10^{-4}	1.8878×10^{-4}
Level 3	1.9489×10^{-4}	1.9436×10^{-4}
Level 4	2.0038×10^{-4}	1.9410×10^{-4}
Level 5	1.9491×10^{-4}	1.9294×10^{-4}
Level 6	1.9396×10^{-4}	1.9658×10^{-4}

The results for the first set of validation scenarios with Gaussian noise (SNR medium) are shown in Table 4.25.

Table 4.25: The fitness values for the first set of validation scenarios with Gaussian noise (SNR medium).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.6367×10^{-4}	1.8130×10^{-4}
Level 2	1.8411×10^{-4}	1.8112×10^{-4}
Level 3	1.9286×10^{-4}	1.8530×10^{-4}
Level 4	1.9406×10^{-4}	1.8743×10^{-4}
Level 5	1.8869×10^{-4}	1.8559×10^{-4}
Level 6	1.8675×10^{-4}	1.8841×10^{-4}

The results for the training scenarios with Gaussian noise (SNR low) are shown in Table 4.26.

Table 4.26: The fitness values for the training scenarios with Gaussian noise (SNR low).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.4223×10^{-4}	1.5583×10^{-4}
Level 2	1.5777×10^{-4}	1.5716×10^{-4}
Level 3	1.5142×10^{-4}	1.5969×10^{-4}
Level 4	1.5613×10^{-4}	1.5620×10^{-4}
Level 5	1.5232×10^{-4}	1.5237×10^{-4}
Level 6	1.5149×10^{-4}	1.5473×10^{-4}

The results for the first set of validation scenarios with Gaussian noise (SNR low) are shown in Table 4.27.

Table 4.27: The fitness values for the first set of validation scenarios with Gaussian noise (SNR low).

	Fitness (for agent with sequence depth 1)	Fitness (for agent with sequence depth 2)
Level 1	1.3859×10^{-4}	1.5147×10^{-4}
Level 2	1.5233×10^{-4}	1.5239×10^{-4}
Level 3	1.4618×10^{-4}	1.5437×10^{-4}
Level 4	1.5066×10^{-4}	1.5059×10^{-4}
Level 5	1.4679×10^{-4}	1.4650×10^{-4}
Level 6	1.4572×10^{-4}	1.4859×10^{-4}

4.5. Remarks

When the results for the training and the validation scenarios are examined, it is apparent that the proposed HDM model operates well. At hierarchical levels generally, incremental changes in the fitness values are observed. However, when noise is added, the proposed HDM model shows poor performance in the experiments with low SNRs. However, for high SNR situations, the performance indices are still acceptable. From a lower level to an upper level generally there is an increase in the fitness values. So there is a distinction between the way the HDM model is applied in Chapter 3 and in Chapter 4. The results in these chapters show different noise resistance characteristics, especially when the SNR is high. The main reason for this situation could be because of the clustering method and distance-based membership degree assignment process for the sequences in the rule-base. The membership degree of a sequence (for a sample) is determined according to a cumulative distance measure between the samples in the window and the centers of clusters available in the sequence. When noise is applied, the cumulative distance measure may not be affected much, since it is determined by a number of clusters (different reference points) in the sequence. Thus, the distance-based membership degree assignment process (i.e., pre-processing the data before directly feeding it into the rule-base) decreases the accumulation of decision error at the hierarchical levels due to the fusion equation.

CHAPTER 5

LOCAL DECISION-MAKING IN MULTIPLE-LEVELS

The multi-level DM model proposed in this chapter, "local DM in multiple-levels" (LDM-ML), has a decentralized and distributed nature. The main idea is to develop local agents to make decisions in the local regions to which they are assigned. The decisions of local agents are fused using a suitable decision-fusion method to determine all the decisions in a level. The success of a local agent is calculated based on a user-provided performance measure. If an agent is successful, it is also used for the determination of decisions at the next level. Otherwise it is eliminated from the agent set and new candidate local agents are developed at the next level instead of the eliminated local agent. The success of the model is measured according to an overall performance criterion. This model is compared with some known techniques over CS1 (the Bayes Optimal Classifier in Chapter 5 and the Adaboost in Chapter 6).

5.1. Application

CS1 defined in Chapter 2 and analyzed in Chapters 3 and 4 is used to develop the second multi-level DM model. Basically, the model is applied to determine the faults in the first tank. The proposed approach and the details of the application are given below:

- The error data is clustered.
- A local decision maker (agent) is developed for each cluster -- an agent developed for a cluster decides on the amount of fault in the selected tank for each sample belonging to the cluster. The development process for an agent mainly concentrates on the samples that are close to the cluster center. This condition is ensured due to a suitably selected cost

function. At the same time, the samples away from the cluster center (but still inside the cluster) have less importance in the development process.

- After the local agents are trained, their decisions are fused using a decision-fusion method. At each level, the same decision-fusion method is applied. The decision about a sample is determined by the local agent (decision maker) developed for the cluster including that sample (i.e., a sample belongs to a cluster if its distance to that particular cluster center is minimum).
- Depending on some performance criteria, when passing from the current level to the next level, the successful decision makers are retained while unsuccessful decision makers are removed from the agent set.
- The samples inside the clusters for which unsuccessful local agents are developed are re-clustered. For each new cluster obtained as a result of re-clustering, new local agents are developed using the principles explained in previous items. The decision-fusion is carried out similarly as in the previous levels.
- New levels are constructed until the performance of the DM model is satisfactory.

5.1.1. Level 1

The computations in the model start with clustering the samples. The error data obtained for the training scenarios is clustered into a relatively small number of clusters (i.e., 25) using the k-means algorithm. For each cluster, a local agent is developed. To develop a local agent, only the samples inside the corresponding cluster are considered. A local agent is a multi-layer perceptron type NN having 3 layers (input layer, hidden layer and the output layer). The activation function for all the neurons is:

$$o' = \frac{2}{\pi}\arctan(\frac{1}{2}o), \tag{5.1}$$

where, 'o' is the input to the neuron, and 'o'' is the output obtained using the activation function.

The number of hidden layer neurons designates the number of weights to be used between the input and the hidden layer and between the hidden layer and the output layer. It directly influences the computation capacity of the agent. There is a trade-off in utilizing a low or a high number of hidden layer neurons in the NN structure. If the number of parameters of the NN (i.e., weights and biases) is relatively large compared to the number of samples used to train the NN, there is a possibility that the NN will memorize the samples and over-fit the given data. Consequently, the generalization ability of the network is diminished. In such a case, the performance of the NN for the training scenarios will be too high; however the performance for the validation scenarios will be inadequate. Conversely, if the number of hidden layer neurons is scarce, it is possible to observe under-fitting in training. In such a case, the performances for both the training and the validation scenarios are poor. Hence, determination of the number of hidden layer neurons is a demanding process where, most of the time, trial and error methods are necessary to reveal the optimal number of hidden layer neurons. For a local agent training simulation, different numbers of hidden layer neurons are tried and the NN relatively successful for both the training and the first set of validation scenarios is chosen as the local agent.

In order to train local agents, the cost function that concentrates mainly on the cluster members (close to center) is chosen to be:

$$Cost_i = \sum_{j=1}^{num\_dat_i} e^{\frac{-\left\| c_{i,center} - x(j) \right\|^2}{\sigma^2}} \left(y(j) - t(j) \right)^2, \tag{5.2}$$

where, '$Cost_i$' is the cost of the NN developed for the cluster 'i'. 'num_dat_i' is the number of samples involved by the corresponding cluster 'i', '$c_{i,center}$' is cluster center (or a focal point) of cluster 'i', 'x(j)' is a sample inside the cluster, 'y(j)' is the normalized fault prediction amount for the sample 'x(j)', and 't(j)' is the real normalized fault amount (i.e., target) in the tank for the sample 'x(j)'. The exponential weight ensures the minimization of the cost at and around the corresponding cluster center. The term 'σ' determines the rate of decay in the exponential weight. 'σ' is chosen comparably small to guarantee the minimization focus directly around the cluster center. In this application, 'σ' is chosen as 0.5.

5.1.1.1. Choice of the Cost Function

As new clusters are introduced into new levels, they may steal some samples belonging to old clusters where successful agents are developed previously. In such a case, the sizes of the old clusters diminish. Generally, the stolen samples are in the regions close to boundaries of the old clusters (i.e., away from the central regions of old cluster centers). Due to this fact, for a specific cluster configuration, the training samples closer to the cluster center are more important (i.e., an old cluster is more likely to lose far away samples when new cluster centers are introduced in the fusion equation). This is the main advantage of choosing such a cost function for training local agents.

5.1.1.2. Optimization Technique and the Decision-Fusion Method

The back-propagation algorithm is used to train the NN with respect to the cost function defined by Equation (5.2). The back-propagation algorithm is performed for 12000 steps, and the step size is adjusted after every 5 steps by a suitable one-dimensional search method. Eventually, 25 local decision agents are developed at the first level. These agents are responsible for the predictions of the faults for the samples inside the cluster they are developed for. The decision-fusion method depends on the spatial distance between the cluster centers and the samples. If a

sample belongs to a cluster, its decision is uniquely determined by the agent developed for the cluster. The decision fusion technique can be summarized as:

$$\min_{i=1}^{n}(\|x(j)-c_{i,center}\|)=\|x(j)-c_{k,center}\|\rightarrow c_{k,center}\in c_k \Rightarrow d_{final,x(j)}=d_{k,x(j)},$$

(5.3)

where, 'n' is the number of cluster, 'x(j)' is the sample, '$c_{k,center}$' is the k^{th} cluster center, 'c_k' is the cluster which includes the k^{th} cluster center, '$d_{final,x(j)}$' is the final decision for the sample 'x(j)' in the level, and '$d_{k,x(j)}$' is the decision of agent developed for cluster 'c_k' for the sample 'x(j)'.

5.1.1.3. Performance Criterion for Fusion Method:

In order to estimate the success of the decision-fusion technique, a cost and a performance measure are defined as:

$$Total\_cost=\sum_{j=1}^{num\_all\_samp}\left|t(j)-d_{final,x(j)}\right|, \quad (5.4)$$

$$Performance=1-\frac{Total\_cost}{2\times num\_all\_samp}, \quad (5.5)$$

where, 'Total cost' is the total absolute cost over all the samples, 't(j)' is the real normalized fault amount for the sample 'x(j)', '$d_{final,x(j)}$' is the final fault prediction (decision) for the sample 'x(j)' in the level, and 'num_all_samp' is the total number of samples in the error data. In order to find the performance of the fusion technique 'Total cost' is divided by '2×num_all_samp' which is the maximum cost value accounting all the samples.

5.1.1.4. Results for the First Level

- Number of samples trained: 34170.
- Total_cost (training) = 2798.5.
- Performance (training) = 0.9590.
- Total_cost (validation) = 3567.6.
- Performance (validation) = 0.9478.

5.1.2. Levels

In the development of new decision makers for the levels, the following items are considered:

- In the fusion process, if a decision maker is relatively unsuccessful (either poor performance values are observed in the training scenarios or in the validation scenarios or in both), it is replaced by a number of new candidate decision makers. In order to determine the successful and unsuccessful agents of any level, the individual performance of the agents for their corresponding clusters are calculated both for the training scenarios and the first set of validation scenarios, using:

$$Per\_of\_age_i = 1 - \frac{\sum_{j=1}^{num\_dat_i} |y(j) - t(j)|}{2 \times num\_dat_i}, \tag{5.6}$$

In Equation (5.6), 'Per_of_age$_i$' is the individual performance of the agent considering all samples inside its corresponding cluster 'i', 'num_dat$_i$' are the number of samples inside the cluster 'i', 'y(j)' is the output of NN for the sample 'x(j)' and 't(j)' is the real fault amount for the sample. The local agents of any level which have an individual performance value comparably small for both data sets (training and first set of validation data sets) or demonstrate drastic differences

between the two data sets are declared to be unsuccessful while the others are declared to be successful.

- The samples inside the cluster, for which the local decision maker failed to yield good individual performance, are re-clustered.
- If the samples of a specific old unsuccessful cluster are re-clustered, it is probable that the new clusters will also include (steal) some samples from the neighboring clusters (they may steal samples from the successful clusters as well).
- The same cost function is used in the optimization of local agents for new clusters. Due to re-clustering, only samples close to the old successful cluster center are retained in the old clusters while samples at the remote boundaries are generally distributed between new clusters. As a result, the samples that are further away and have less dominance for the cost function are transferred to new clusters where new agent training simulations are carried out.

Equation (5.6) is used as a basis to determine the successful and unsuccessful agents in a level. An agent successful at one level may become unsuccessful at the next level in some cases (when successfully classified samples are stolen by other clusters, due to a decrease in the number of samples). However, due to the way in which the cost function is established, it is generally expected that the performance of the agent is augmented at least for the training scenarios. In case of a performance disorder for a successful agent at the following level (decrease in performance for the validation scenarios), that agent is also replaced by a few new agents at the new level.

In the second and the third levels, if a new cluster is generated, it is intended to contain a moderate amount of samples (between 200-400 samples for each cluster). However, due to an uneven distribution of samples and due to the application of the k-means algorithm, adjusting the number of samples in new

clusters precisely is impossible. Hence, the new clusters may possess different amounts of samples after re-clustering.

In the third level, it is observed that some of the unsuccessful clusters have a moderate number of samples (10-200). For this reason, the number of new clusters to be created in place of an unsuccessful cluster is decided to be at most 3 for the fourth level.

5.1.2.1. The Results for the Second Level

- Number of agents used: 65.
- Number of new agents inserted into the agent set: 40.
- Number of samples retrained: 29013.
- Number of agents which have insufficient performance values in the first level and which are re-trained according to new cluster distribution due to creation of new clusters in the second level: 11.
- Total number of agents trained or re-trained: 51.
- Total_cost (training) = 2516.5.
- Performance (training) = 0.9632.
- Total_cost (validation) = 3406.
- Performance (validation) = 0.9502.

5.1.2.2. The Results for the Third Level

- Number of agents used: 125.
- Number of new agents inserted into the agent set: 60.
- Number of agents which have insufficient performance values in the second level and which are re-trained according to new cluster distribution due to creation of new clusters in third level: 6.
- Total number of agents trained or re-trained: 66.

- Number of samples re-trained: 13359.
- Total_cost (training) = 2368.9.
- Performance (training) = 0.9653.
- Total_cost (validation) = 3405.2.
- Performance (validation) = 0.9502.

5.1.2.3. The Results for the Fourth Level

- Number of agents used: 211.
- Number of new agents inserted into the agent set: 86.
- Number of agents which have insufficient performance values in the third level and which are re-trained according to new cluster distribution due to creation of new clusters in fourth level: 29.
- Total number of agents trained or re-trained: 115.
- Number of samples re-trained: 11876.
- Total_cost (training) = 2294.3.
- Performance (training) = 0.9664.
- Total_cost (validation) = 3336.9.
- Performance (validation) = 0.9512.

As observed from the results, there is a continuous decrease in the 'Total cost' value both for the training scenarios and the first set of validation scenarios when moving from a lower level to a higher level. After the agents are developed using the training scenarios and the first set of validation scenarios together, their performance in the second set of validation scenarios are monitored. The results for the second set of validation scenarios are shown in Table 5.1.

Table 5.1: The performance results for the second set of validation scenarios.

	Level 1	Level 2	Level 3	Level 4
Total_cost	2780.2	2533.6	2429	2382.6
Performance	0.9593	0.9629	0.9645	0.9651

In Fig. 5.1, the normalized fault predictions for a scenario is plotted at each level when the developed local agents at each level are used as the decision makers.

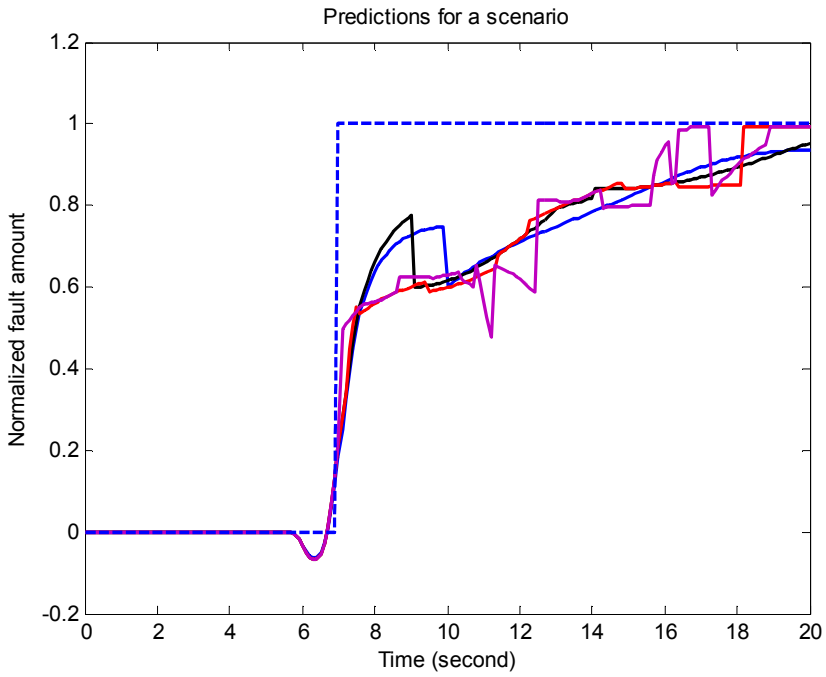

Fig. 5.1: The normalized fault predictions for a scenario when the developed local agents at each level are used as the decision makers. Dotted blue shows the real fault amount, blue shows the first level predictions, black shows the second level predictions, red shows the third level predictions, violet shows the fourth level predictions.

5.2. Comparison by the Bayes Optimal Classifier

A model Bayes optimal classifier [48] is established for comparing the effectiveness of the LDM-ML model. In order to make the classifier operate properly and efficiently, it is necessary to identify the hypotheses and the possible classifications for CS1. For CS1, the hypotheses are the clusters. So if a sample is encountered, one can make a conclusion about the state belonging to the sample (identify precisely the cluster that the sample belongs to). The possible classifications of the sample are the seven different levels of hole deviations. In order to construct the Bayes optimal classifier, it is required to determine the priori probabilities for the hypotheses and posteriori probabilities for the classifications. The probabilities are defined below:

The priori probability (i.e., $P(c_i/x)$) is the probability of the occurrence of cluster 'c_i' given that sample x is observed. The posteriori probabilities for the classifications are also necessary to construct the Bayes optimal classifier. 'When a cluster is observed, what is the probability of a classification' is the question to be answered (i.e., $P(v_j/c_i)$ values where 'v_j' is the j^{th} type classification and 'c_i' is the i^{th} cluster).

There are many ways to measure the posteriori and priori probabilities. Generally a measure of distance between samples and cluster centers is a good way to determine the priori probabilities. Unfortunately, this distance measure may also possess some variants. Through different efforts to determine the priori probabilities, the best one is obtained using Equation (5.7)

$$\arg(\min_{i=1}^{n}\left\|c_{i,center}-x\right\|)=c_{k,center}\Rightarrow P(c_k/x)=1,\ \sum_{i=1,i\neq k}^{n}(c_i/x)=0. \qquad (5.7)$$

In order calculate the posteriori probabilities, Equation (5.8) is used which exploits the statistics of the training samples:

$$P(v_j/c_i)=\frac{num((samples\_in\_c_i)\cap(samples\_in\_v_j))}{num(samples\_in\_c_i)}. \qquad (5.8)$$

In Equation (5.7), 'c_i' represents i^{th} cluster, '$c_{i,center}$' represents the cluster center of 'c_i', 'x' represents the observed sample, 'n' represents the number of clusters available in the considered level. In Equation (5.8), 'num(samples_in_c_i)' represents the number of samples involved in the cluster 'c_i'. 'num((samples_in_c_i)∩(samples_in_v_j))' represents the number of samples which are both elements of cluster 'c_i' and simultaneously classified as the type 'v_j'. Both priori and posteriori probabilities are determined using the statistics of the error data since they are created artificially by us. Using these probabilities, the Bayes optimal classifier determines the classification of a sample using:

$$P(v_k, x) = \arg\max_{v_j \in V} \left(\sum_{i=1}^{n} P(v_j / c_i) P(c_i / x) \right) \Rightarrow x \in v_k . \qquad (5.9)$$

where, 'V' represents the set of all possible attribute classifications: A sample can be classified as having positive big (PB), positive medium (PM), positive small (PS), zero (Z), negative small (NS), negative medium (NM) and negative big (NB) normalized fault characteristics.

A procedure to compare the results of the Bayes optimal classifier with the results of the proposed DM model is necessary. For this purpose, the classifications of the Bayes optimal classifier for each training sample should be converted into numerical values. This procedure helps the calculation of the 'Total_cost' and 'performance' (i.e., Equation (5.4) and Equation (5.5)) for the Bayes optimal classifier. The transformation of the sample classifications is as follows:

- If a sample is classified as 'NB', the normalized prediction is -1.
- If a sample is classified as 'NM', the normalized predicted is -0.66.
- If a sample is classified as 'NS', the normalized predicted is -0.33.
- If a sample is classified as 'Z', the normalized predicted is 0.
- If a sample is classified as 'PS', the normalized predicted is 0.33.
- If a sample is classified as 'PM', the normalized predicted is 0.66.
- If a sample is classified as 'PB', the normalized predicted is 1.

The results for the training scenarios are shown in Table 5.2

Table 5.2: Results for the Bayes optimal classifier (for the training scenarios).

	Level 1 (25 clusters)	Level 2 (65 clusters)	Level 3 (125 clusters)	Level 4 (211 clusters)
Total_cost	6958.2	5368.2	4270.6	3590.
Performance	0.8982	0.9214	0.9375	0.9475

In order to utilize the Bayes optimal classifier for the validation scenarios, the priori probabilities for the validation scenarios is re-determined using Equation (5.7). However, the posteriori probabilities for classifications are taken the same values as previously calculated in training scenarios using Equation (5.8). But at each level the posteriori probabilities are recalculated and updated for training scenarios since new clusters are introduced in the Bayes optimal classifier classification equation. Equation (5.9) is used as the Bayes optimal classifier classification method. The results for the first set of validation scenarios are shown in Table 5.3.

Table 5.3: Results for the Bayes optimal classifier (the first set of validation scenarios).

	Level 1 (25 clusters)	Level 2 (65 clusters)	Level 3 (125 clusters)	Level 4 (211 clusters)
Total_cost	7312.9	5607.8	4638.9	4508.1
Performance	0.8930	0.9179	0.9321	0.9340

As seen from the results, although the transformation is in favor of the Bayes optimal classifier, the performance values for the proposed DM model are better.

If the Bayes optimal classifier is to classify a sample correctly, its contribution to the total cost is 0. On the contrary, the contribution of the proposed DM model to the total cost for the same sample is non-zero most of the time, even if the decision about the fault amount for the sample is confidential enough. For this reason, the transformation is always in favor of the Bayes optimal classifier. Besides there is a second advantage of the Bayes optimal classifier that originates from the re-adjustment of the '$P(v_j,c_i)$' values (the posteriori probabilities) for each cluster at each level over again. Due to the redistribution of the samples between old clusters and new clusters introduced in each level, the posteriori probabilities for classifications are recalculated afresh at every level that contributes a further improvement in 'Total_cost' and 'performance' values. However, in the proposed DM model, such a facility does not exist. Most of the time the decision about a sample remains the same if that sample is not involved with a new created cluster in the forthcoming levels.

5.3. Noise

Two different applications are performed with noise. Performance of the Bayes optimal classifier and the LDM-ML model are compared in terms of endurance to noise. To realize this comparison, the noisy data generated using a uniform distribution with a different SNR settings (high, medium law) is used (the details are given in Chapter 2). The second application is performed in order to observe the effect of selecting different distributions for noise (Gaussian distribution, the detail are given in Chapter 2): It is desired to learn if there is any drastic difference in performance values if a different noise distribution is chosen.

5.3.1. Application 1: Comparison of the Proposed Model with the Bayes Optimal Classifier for Uniform Noise

Utilizing the noisy data created using the uniform distribution, the performance of the proposed local DM model and the Bayes optimal classifier are compared in terms of noise endurance (durability). In the same way, the same procedure is carried out for the first set of validation scenarios. The results are shown in Tables 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, and 5.11 respectively.

Table 5.4: The performance of the first level local agents (25 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 1	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9474	0.8956	0.8513
Performance (validation)	0.9357	0.8840	0.8413

Table 5.5: The performance of the second level local agents (65 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 2	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9496	0.8930	0.8453
Performance (validation)	0.9364	0.8814	0.8361

Table 5.6: The performance of third level local agents (125 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 3	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9517	0.8954	0.8530
Performance (validation)	0.9365	0.8824	0.8440

Table 5.7: The performance of the fourth level local agents (211 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 4	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9535	0.9010	0.8659
Performance (validation)	0.9382	0.8882	0.8573

Table 5.8: The performance of the Bayes optimal classifier (25 cluster case) for noisy data (both for the training and the first set of validation scenarios).

25 clusters	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.8982	0.8978	0.8973
Performance (validation)	0.8931	0.8933	0.8933

Table 5.9: The performance of the Bayes optimal classifier (65 cluster case) for noisy data (both for the training and the first set of validation scenarios).

65 clusters	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9214	0.9205	0.9173
Performance (validation)	0.9180	0.9173	0.9150

Table 5.10: The performance of the Bayes optimal classifier (125 cluster case) for noisy data (both for the training and the first set of validation scenarios).

125 clusters	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9373	0.9329	0.9156
Performance (validation)	0.9319	0.9279	0.9100

Table 5.11: The performance of the Bayes optimal classifier (211 cluster case) for noisy data (both for the training and the first set of validation scenarios).

211 clusters	(SNR high)	(SNR medium)	(SNR low)
Performance (training)	0.9471	0.9339	0.9058
Performance (validation)	0.9337	0.9221	0.8961

5.3.2. Application 2: Comparison of the Proposed Model with the Bayes Optimal Classifier for Gaussian Noise

Utilizing the noisy data created using Gaussian distribution, the performance of the proposed local DM model and the Bayes optimal classifier are compared in terms of noise endurance (durability). In the same way the same procedure is carried out for the first set of validation scenarios. The results are shown in Tables 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18, and 5.19 respectively.

Table 5.12: The performance of the first level local agents (25 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 1	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9480	0.9025	0.8586
Performance (validation)	0.9363	0.8914	0.8413

Table 5.13: The performance of the second level local agents (65 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 2	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9503	0.9002	0.8542
Performance (validation)	0.9372	0.8890	0.8456

Table 5.14: The performance of third level local agents (125 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 3	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9523	0.9026	0.8607
Performance (validation)	0.9371	0.8901	0.8519

Table 5.15: The performance of the fourth level local agents (211 agent case) for noisy data (both for the training and the first set of validation scenarios).

Level 4	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9541	0.9075	0.8694
Performance (validation)	0.9388	0.8945	0.8611

Table 5.16: The performance of the Bayes optimal classifier (25 cluster case) for noisy data (both for the training and the first set of validation scenarios).

25 clusters	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.8982	0.8982	0.8967
Performance (validation)	0.8932	0.8936	0.8928

Table 5.17: The performance of the Bayes optimal classifier (65 cluster case) for noisy data (both for the training and the first set of validation scenarios).

65 clusters	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9212	0.9211	0.9146
Performance (validation)	0.9179	0.9180	0.9124

Table 5.18: The performance of the Bayes optimal classifier (125 cluster case) for noisy data (both for the training and the first set of validation scenarios).

125 clusters	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9370	0.9327	0.9149
Performance (training)	0.9319	0.9276	0.9108

Table 5.19: The performance of the Bayes optimal classifier (211 cluster case) for noisy data (both for the training and the first set of validation scenarios).

211 clusters	SNR (high)	SNR (medium)	SNR (low)
Performance (training)	0.9470	0.9346	0.9071
Performance (validation)	0.9340	0.9230	0.8971

When the results are compared, the LDM-ML model works pretty well if the SNR is high, but its performance diminishes when the noise amount is increased. When moving from a lower level to higher level, there is an increase in the performance

values for both the training and the validation scenarios no matter which noise distribution is selected; the same situation is not observed for the Bayes optimal classifier. For low and medium SNR situations, the Bayes optimal classifier is better. However, when moving from a lower level to a higher level, there may be a drastic decline in performance, which is undesired. There may be several reasons for the encountered performance decline for the Bayes optimal classifier. For the first or the second levels, the number of clusters is scarce. Due to the way in which the priori probabilities are determined, every piece of data inside a cluster is classified to have the same type of classification. As the number of clusters are increased, the area the clusters occupy decreases. When noise is also added to the data, due to the decreased cluster size, the possibility for the noisy data to penetrate to another cluster increases. Thus, after some critical value, if the number of clusters are further increased, the performance decreases.

5.4. Remarks

The main advantage of LDM-ML model is that the number of examples (samples of error data) that should be retrained in training scenarios decreases from level to level since agents that are sufficient enough in previous levels are kept in the successful agent set so that re-use of the previously classified data is provided if the related agent is successful. This has a positive effect to diminish the computation time required to train the new agents at each level. Secondly, as seen from the performances values in both the training and validation data, the model outperforms the Bayes optimal classifier although at each level the Bayes optimal classifier is retrained to determine the posteriori probability values.

As noise endurance characteristics are compared, the LDM-ML model is better when the SNR is high whereas when the SNR decreases the Bayes optimal classifier demonstrates better performance values. However, increasing the number of clusters above some critical value has a spoiling effect for the Bayes optimal classifier when the SNR is medium or low which does not seem to create a problem for the LDM-ML model.

One of the drawbacks of the proposed model is the way to determine the new clusters. Clustering techniques different from the k-means algorithm can be used to create more manageable clusters to train agents, but this procedure necessitates some priori information about the samples which we lack. One of the difficulties in the applied model is the adjustment of the number of neurons in the hidden layer (i.e., in the same way the number of weights to be optimized). Even the structure of the NN may be questionable in some cases (the activation function, number of hidden layers used in the NN, the number of one dimensional search iterations necessary to prevent under-fitting and over-fitting). But these problems originate due to the parameters of the applied model. Tuning all of these parameters to obtain the perfect solution is nearly impossible since the structures of each agent at every level should be re-examined again and again. Lastly, one of the problems encountered due to the use of proposed DM model is for the determination of the dimensions of the regions (clusters), the agents are influential. Due to the trade-off between the performances in the training and validation scenarios, this issue may become a matter. Sometimes there is an uneven distribution of samples in a cluster for the training and the validation scenarios. The number of training samples in a cluster can be scarce compared to the validation samples, leading to poor performance for validation samples. Two ideas can be used to prevent this problem. First, the size of the cluster can be increased such that some extra safety region at the boundaries of cluster can also be occupied by the cluster. This will increase the size of the cluster. Hence more training samples will be included in the cluster, but, in that case, due to increased number of samples in the cluster training, an agent requires more computation time. The second idea is to increase the number of training scenarios so that extra training samples will be inserted into the clusters. However, the application of the second idea also increases the computation time to train the agents. Another difficulty is that determining the safety regions may have a negative effect and decrease the efficiency of the agents created.

CHAPTER 6

THE ADABOOST

In order to test the effectiveness of developed DM models, an Adaboost classifier has been decided to be constructed. Two different applications are carried out depending on the formation of the base classifiers. In the first application, the base classifiers are selected as the local agents developed for LDM-ML model at Chapter 5. In the second application, 35 different base classifiers are created using the HDM model as explained and evaluated similar to Chapter 4 and the outputs of these base classifiers are fused using the Adaboost algorithm.

Usually, Adaboost is used for classification of data into two classes. However with some modification, it can be used to classify data into multiple classes. One of the approaches performed for multi-class Adaboost is demonstrated at [49]. A similar Adaboost algorithm is also used in this chapter.

6.1. Algorithm

6.1.1. Initialization

- Let $L = \{(x_i, t_i)\}_{i=1}^{N}$ be a training set, where N is the number of training examples,
- $x_i \in X \subset R^m$ be the input data of the training set,
- $t_i \in T = \{-1,1,-0.66,0.66,-0.33,0.33,0\}$ be the targets associated for the input data,
- $h_m(.)$ be a weak classifier such that $h_m(x_i) \in T$,
- M be number of weak classifiers.

6.1.2. Initialization of Probability Density Function

Set the probability density function over L such that $D_1(i) = 1/N$, for i=1,2,3,...,N.

6.1.3. Iteration:

For m = 1 to M,

a) Train $h_m(.)$ using a weak learning algorithm.

b) Compute the error of h_m:

$$Err^m = \sum_{i=1}^{N} I(h_m(x_i) \neq t_i) \times D_m(i). \quad (6.1)$$

In this error function 'I(.)' is the indicator function such that if $h_m(x_i) \neq t_i$ then its output is one, otherwise (i.e., if $h_m(x_i) = t_i$) its output is 0.

c) Compute α^m such that

$$\alpha^m = \log\left(\frac{1 - Err^m}{Err^m}\right) + \log(K - 1). \quad (6.2)$$

In this function K shows the number of possible classifications. For the related problem with K = 7, there are 7 different possible classifications.

d) Update the probability density function over L:

$$D_{m+1}(i) = \frac{D_m(i)}{Z_m} \times \left\{ \begin{matrix} e^{-\alpha^m}, if & h_m(x_i) = t_i \\ e^{\alpha^m}, if & h_m(x_i) \neq t_i \end{matrix} \right\}. \quad (6.3)$$

where, 'Z_m' is the normalization factor such that 'D_{m+1}' is also a probability density function for the training examples.

End of For loop

6.1.4. Output of Ensemble Classifier:

$$C(x_i) = \arg\max_k \left(\sum_{m=1}^{M} \alpha^m \times I(h_m(x_i) = k) \right), \qquad (6.4)$$

where, 'I(.)' is the indicator function such that if $h_m(x_i) = k$, then the output for indicator function is 1, whereas the output of indicator function is 0 if $h_m(x_i) \neq k$.

6.2. Application 1

In the first application, the weak classifiers are selected as the local agents of Chapter 5. These agents are assigned as the weak classifiers of the Adaboost algorithm. Given an input data these classifiers yields a numeric output value between 1 and -1. One has to transfer the output value to an attribute value and the attribute value should be also transferred to a numerical normalized prediction. The mapping for the transfer is carried according to following rules:

If output of the weak classifier is in:

- [-1, 0.833), the fault in the tank is 'NB' and the normalized prediction is -1.
- [-0.833, 0.5), the fault in the tank is 'NM' and the normalized predicted is -0.66.
- [-0.5, -0.167), the fault in the tank is 'NS' and the normalized predicted is -0.33.
- [-0.167, 0.167], the fault in the tank is 'Z' and the normalized predicted is 0.
- (0.167, 0.5], the fault in the tank is 'PS' and the normalized predicted is 0.33.
- (0.5, 0.833], the fault in the tank is 'PM' and the normalized predicted is 0.66.
- (0.833, 1], the fault in the tank is 'PH' and the normalized predicted is 1.

Using this transformation, the outputs for each weak classifier are obtained. Then using the Adaboost algorithm the outputs of the weak classifiers are aggregated. The performance (using Equation (5.5)) and 'Total_cost' (using Equation (5.4)) values obtained as the result of algorithm for each level are shown in Table 6.1:

Table 6.1: The performance and total cost results obtained using the Adaboost algorithm where base classifiers are selected as the local agents developed at Chapter 5 at each level.

	Level 1	Level 2	Level 3	Level 4
Performance	0.9327	0.9032	0.9063	0.8943
Total _cost	4602.6	6613	6406.1	7226

When compared to the LDM-ML applied in Chapter 5, the Adaboost algorithm show worse performance values while passing from a lower level to an up level.

6.3. Application 2

In the second application to test the effectiveness of the Adaboost algorithm, 35 different base classifiers (agents) are developed which possess the rule structure explained in Chapter 4 (the agents are made of rule bases which are composed of sequences). The evaluation of the rule bases is the same as the evaluation strategy explained in Chapter 4. Each agent are developed with different sequence depth and sequence length settings, and in the development of each agent different number of cluster centers are utilized at different simulations (sequence depth is either selected as 1 or 2, sequence length is selected as 5, 10, 15 or 20 and the number of clusters available in the optimization process are either 10 or 20 or 30 or 40 or 50. Each developed agent has a different setting among these parameters). In the development process GA is used as in chapter 4. In GA agent development simulations, the population size is determined as 40 whereas the number of

generations is selected as 300. As the agents are developed, the transformation used in the first application of this Chapter is applied to the outputs of the agents. Consequently, the base classifiers are obtained. Using the Adaboost algorithm, the outputs of base classifiers are aggregated and the corresponding performance values are obtained:

- Performance = 0.9272.
- Total_cost = 4977.8.

This shows that the Adaboost is not as effective as the proposed model in the previous applications (in Chapter 4) when the training results are compared.

CHAPTER 7

CONVERGENCE ANALYSIS OF MODELS

In this chapter the convergence characteristics of the proposed multi-level DM models have been studied. These studies demonstrate that under certain conditions and a-priori assumptions about the model structures, it is possible to improve the performance measure of the DM models as the levels are increased. The first study is about LDM-ML model as applied in Chapter 5 and the second study is about the HDM model as applied in Chapter 3. The second study can be expanded to cover the HDM model with multiple agent structures discussed in Chapter 4, as well. However these studies are done depending on some a-priori assumptions. Besides some components of the DM models (i.e., construction and the use of performance plots, clustering style for determining local clusters, determination of a focal point 'cluster center' in a cluster) are somewhat simplified and slightly modified without loss of generality, in order to simplify the proofs. These two studies show to what extent the proposed methods are successful.

There are also some sections of this chapter which explain how these proofs can be generalized. These sections mainly stress the strategies to deal with problematic parts of the theoretical studies that are not modeled in the proofs (i.e., over-fitting and under-fitting situations when NNs are used, the use of a set of validation data and training data together to construct the local DM model - which is not emphasized in the theoretical study).

7.1. Theoretical Study 1: Local Decision-Making in Multiple-Levels (LDM-ML)

7.1.1. The Model

This section is about the LDM-ML model in Chapter 5. Let $r_{1,x},\ldots, r_{k,x}$ be the clusters (regions), $c_{1\_center,x}$, $c_{2\_center,x}$, …, $c_{k\_center,x}$ be the associated centers of each cluster, $a_{1,x},\ldots, a_{k,x}$ be the associated agents for regions $r_{1,x},\ldots, r_{k,x}$, $num_{1,x},\ldots, num_{k,x}$ be the number of data points possessed by the regions $r_{1,x},\ldots, r_{k,x}$, $p_{1,x},\ldots, p_{k,x}$ be the performance values of agents $a_{1,x},\ldots, a_{k,x}$ at level 'x', respectively. The cost function for training '$a_{i,x}$' is defined as

$$Cost_{i,x} = \sum_{j=1, d(j)\in r_{i,x}}^{num_{i,x}} e^{\frac{-\left\| c_{i\_center,x} - d(j) \right\|^2}{\sigma^2}} (y(j) - t(j))^2, \qquad (7.1)$$

where 'd(j)' is any input data point possessed by the region '$r_{i,x}$', 'y(j)' is the output (decision) determined by '$a_{i,x}$' for data 'd(j)', and 't(j)' is the output that should be obtained (target) for data 'd(j)'. The performance of agent '$a_{i,x}$' is calculated by

$$p_{i,x} = 1 - \frac{\sum_{j=1, d(j)\in r_{i,x}}^{num_{i,x}} \left| y(j) - t(j) \right|}{(t_{\max} - t_{\min}) \times num_{i,x}}, \qquad (7.2)$$

where, 't_{max}-t_{min}' term represents the difference between the maximum value and the minimum value that can be assigned to a target. This term guarantees that,

$$0 \le p_{i,x} \le 1.$$

At level ‘x’, the total performance is calculated as

$$P_{total\_level,x} = \frac{\sum_{i=1}^{k} num_{i,x} \times p_{i,x}}{\sum_{i=1}^{k} num_{i,x}} = \frac{\sum_{i=1}^{k} num_{i,x} \times p_{i,x}}{N_T}, \quad (7.3)$$

where ‘N_T’ is the total number of data points in all the regions.

7.1.1.1. General Assumption

The assumption states that in a given region the data is distributed in a relatively fair disposition.

Assumption 1 (no outlier in the region assumption): It is assumed that each cluster contains no outliers in its region:

$$For \quad \forall d(j) \in r_{i,x} \quad where \quad j = 1,...,m,n..,num_{i,x},$$

Given any $\varepsilon > 0$, $\exists\, K(\varepsilon)$ such that whenever $\|d(n) - d(j)\| < \varepsilon \Rightarrow \|t(n) - t(m)\| < K$

This assumption emphasizes that the targets of inputs which are very close to each other with respect to a distance measure should not be much different from each other.

Our aim is to show that as one passes from level ‘x’ to ‘x+1’, the performance measure improves, i.e.,

$$P_{total\_level,x+1} > P_{total\_level,x}. \quad (7.4)$$

7.1.2. Case 1

This case is for the condition when the numbers of hidden layer neurons in the NNs that function as local agents are unlimited. According to the Universal Approximation Theorem [50-51] (see Appendix B), one can train a NN up to any desired performance for each cluster. As a result one can reach a specified performance automatically without a need for increasing the levels.

7.1.3. Case 2

In this case there is an upper bound on the number of neurons and hidden layers of NNs which are to be trained as local agents for each cluster (i.e., $n < n_{critical}$ where $1 \leq n$, and n is the number of hidden layer neurons in the NN and '$n_{critical}$' is the limit value for the number of hidden layer neurons). Since number of hidden layer neurons is limited, Universal approximation theorem is not directly applicable and it is reasonable to have a number of agents in a given level with unsatisfactory performance. It is assumed that the regions associated with successful agents are preserved.

Without loss of generality, let us assume that at level x there is an agent with unsatisfactory performance. This means that for $i = 1,2,\ldots,k-1$, $p_{i,x} \geq M$, whereas $p_{k,x} < M$ (meaning that agents '$a_{i,x}$' i = 1,…,k-1 are successful according to the performance measure given by Equation (7.2) and '$a_{k,x}$' is unsuccessful according to the same measure and '$r_{k,x}$' is the region where unsuccessful agent is associated with).

Performance of the overall model is improved based on the following algorithm:

Initialization Step: Divide the unsuccessful region into 's' sub-regions and go to the main step. Usually 's' is an integer greater than or equal to 2.

Main Step: Unsuccessful region consists of 's' sub-regions ($r_{k,x+1}$, $r_{k+1,x+1}$,….,$r_{k+s-1,x+1}$). The centers ($c_{k\_center,x+1}$, $c_{k+1\_center,x+1}$, $c_{k+2\_center,x+1}$,….., $c_{k+s-1\_center,x+1}$) of $r_{k,x+1}$, $r_{k+1,x+1}$,…., $r_{k+s-1,x+1}$ are determined by k-means clustering in the region '$r_{k,x}$'. Please note that, at least a single data point is exists in each of the new sub-regions.

Step 1: The regions associated with successful agents are preserved. This means that $r_{i,x+1} = r_{i,x}$, $num_{i,x+1} = num_{i,x}$ and as a result $p_{i,x+1} = p_{i,x}$ and the agents '$a_{i,x}$' for i = 1,..,k-1 are preserved.

Step 2: The agent '$a_{k,x}$' is removed from the agent set.

Step 3: New agents are designed associated with the regions $r_{k,x+1},\ldots,r_{k+s-1,x+1}$ in the form of NNs with hidden layers having at most 'n' hidden layer neurons. The number of data points contained in these regions are given by $num_{k,x+1},\ldots.$, $num_{k+s-1,x+1}$ such that,

$$num_{k,x} = \sum_{i=k}^{k+s-1} num_{i,x+1}. \tag{7.5}$$

It is to be reminded that no data points are interchanged between the regions associated with the successful agents (according to the assumption for Case 2).

Step 4: Calculate the performances '$p_{i,x+1}$' for i = k,…, s-1 of the new agents '$a_{i,x+1}$' i = k,…, s-1.

Step 5: Without loss of generality, we can assume that there are two possibilities:

a) Suppose $p_{i,x+1} \geq M, \quad \forall i = 1,..,s-1$, then

$$\begin{aligned} P_{total\_level,x} &= \frac{1}{N_T}\sum_{i=1}^{k} num_{i,x} \times p_{i,x} \\ &= \frac{1}{N_T}\left(\sum_{i=1}^{k-1} num_{i,x} \times p_{i,x} + num_{k,x} \times p_{k,x}\right). \end{aligned} \tag{7.6}$$

$$P_{total\_level,x+1} = \frac{1}{N_T}\sum_{i=1}^{k+s-1} num_{i,x+1} \times p_{i,x+1}$$
$$= \frac{1}{N_T}(\sum_{i=1}^{k-1} num_{i,x+1} \times p_{i,x+1} + \sum_{i=k}^{k+s-1} num_{k,x+1} \times p_{k,x+1}). \quad (7.7)$$

Note that $num_{i,x+1} = num_{i,x}$ and $p_{i,x+1} = p_{i,x}$ i= 1,…,k-1, we also know that $p_{i,x+1} \geq M > p_{k,x}, \quad for\ i = k,...,k+s-1$. These imply that $P_{total\_level,x+1} > P_{total\_level,x}$ (please remember Equations (7.5), (7.7)).

b) Suppose, at least one $p_{i,x+1} < M$, for some $i = k,..,k+s-1$. Without loss of generality, we assume that there is only one bad performance, say '$p_{k+s-1,x+1}$'. Choose a suitable 's' and go to **Main Step**, and apply the algorithm this time to new unsuccessful region '$r_{k+s-1,x+1}$' as previously applied to '$r_{k,x}$'.

Remarks: If there is more than one unsuccessful region, the previous algorithm can be used repeatedly for each unsuccessful region. The essence of the proof in Case 2 depends on the following fact: The regions associated with unsuccessful agents are divided into sub-regions each of which containing a smaller number of data points with respect to the initial regions. Hence, training a NN as a local agent associated with a sub-region has more chance of having sufficiently more performance.

7.1.4. Case 3

In this case there is an upper bound on the number of neurons and hidden layers of NNs which are to be trained as local agents for each cluster (i.e., $n < n_{critical}$ where $1 \leq n$, and 'n' is the number of hidden layer neurons in the NN and '$n_{critical}$' is the limit value for the number of hidden layer neurons). Since number of hidden layer

neurons is limited, Universal approximation theorem is not directly applicable and it is reasonable to have a number of agents in a given level with unsatisfactory performance. It is assumed that the regions associated with successful agents are not preserved when a number of new agents are inserted into the agent set in place of the unsuccessful agents

Without loss of generality, let us assume that at level x there is a single agent with unsatisfactory performance. This means that for $i = 1,2,\ldots,k-1,\ p_{i,x} \geq M$, whereas $p_{k,x} < M$ (meaning that agents '$a_{i,x}$' i = 1,…,k-1 are successful according to the performance measure given by Equation (7.2) and '$a_{k,x}$' is unsuccessful according to the same measure and '$r_{k,x}$' is the region where unsuccessful agent is associated with).

Performance of the overall model is improved based on the following algorithm:

Initialization Step: Divide the unsuccessful region into 's' sub-regions and go to the main step. Usually 's' is an integer greater than or equal to 2.

Main Step: Unsuccessful region is divided into 's' sub-regions ($r_{k,x+1}$, $r_{k+1,x+1},\ldots,r_{k+s-1,x+1}$). The centers ($c_{k\_center,x+1}$, $c_{k+1\_center,x+1}$, $c_{k+2\_center,x+1},\ldots..$, $c_{k+s-1\_center,x+1}$) of $r_{k,x+1}$, $r_{k+1,x+1},\ldots.$, $r_{k+s-1,x+1}$ are determined by k-means clustering in the region '$r_{k,x}$'. However the total area occupied by the sub-regions $r_{k,x+1}$, $r_{k+1,x+1},\ldots,r_{k+s-1,x+1}$ is not equal to the area of '$r_{k,x}$', since some data points can move from the regions associated with successful agents to regions associated with new agents to be developed. Please note that, at least a single data point exists in each of the new sub-regions.

Step 1: The regions associated with successful agents are not preserved. For this reason it is possible that for some i = 1,..,k-1 $r_{i,x+1} \neq r_{i,x}$, $num_{i,x+1} \neq num_{i,x}$ and as a result $p_{i,x+1} \neq p_{i,x}$ although the agents '$a_{i,x}$' for i = 1,..,k-1 are the same.

Even though all the successful regions are not preserved some decisions of level 'x' are preserved since still they are being determined by the successful agents of level 'x'. Let '$r_{preserved,x}$' be the preserved region associated with successful agents of level 'x', '$num_{preserved,x}$' be the number of data points belonging to '$r_{preserved,x}$', '$r_{non\text{-}preserved,x}$' be the non-preserved region associated with successful agents of level 'x' and '$num_{non\text{-}preserved,x}$' be the number of data points belonging to '$r_{non\text{-}preserved,x}$'. There is no doubt that,

$$num_{non-reserved,x} + num_{preserved,x} = \sum_{i=1}^{k-1} num_{i,x}. \tag{7.8}$$

$$r_{preserved,x} \cup r_{non-preserved,x} = \bigcup_{i=1}^{k-1} r_{i,x}. \tag{7.9}$$

The performance for the preserved data points can be calculated as,

$$p_{preserved,x} = 1 - \frac{\sum_{i=1, d(j) \in r_{preserved,x}}^{num_{preserved,x}} |y(j) - t(j)|}{(t_{\max} - t_{\min}) \times num_{preserved,x}}. \tag{7.10}$$

Similarly, the performance for the non-preserved data points can be calculated as,

$$p_{non-preserved,x} = 1 - \frac{\sum_{i=1, d(j) \in r_{non-preserved,x}}^{num_{non-preserved,x}} |y(j) - t(j)|}{(t_{\max} - t_{\min}) \times num_{non-preserved,x}}. \tag{7.11}$$

The total performance at level can be calculated using,

$$P_{total\_level,x} = \frac{p_{preserved,x} \times num_{preserved,x} + p_{non-preserved,x} \times num_{non-preserved,x} + p_{k,x} \times num_{k,x}}{N_T}, \tag{7.12}$$

where,

$$N_T = num_{preserved,x} + num_{non-preserved,x} + num_{k,x}. \tag{7.13}$$

Step 2: The agent '$a_{k,x}$' is removed from the agent set.

Step 3: New agents are designed associated with the regions $r_{k,x+1},\ldots, r_{k+s-1,x+1}$ in the form of NNs with hidden layers having at most 'n' hidden layer neurons. The number of data points contained in these regions are given by $num_{k,x+1},\ldots., num_{k+s-1,x+1}$ such that,

$$num_{non-preserved,x} + num_{k,x} = \sum_{i=k}^{k+s-1} num_{i,x+1}. \tag{7.14}$$

Step 4: Calculate the performances '$p_{i,x+1}$' for i = k,…, s+k-1 of the new agents '$a_{i,x+1}$' i = k,…, s+k-1. Using the performances of new agents, we can write

$$P_{total\_level,x+1} = \frac{p_{preserved,x} \times num_{preserved,x} + \sum_{i=k}^{k+s-1} p_{i,x+1} \times num_{i,x+1}}{N_T}. \tag{7.15}$$

Step 5: Without loss of generality, we can assume that there are two possibilities:

a) Suppose $p_{i,x+1} \geq \dfrac{p_{non-preserved} \times num_{non-preserved} + M \times num_{k,x}}{num_{non-preserved,x} + num_{k,x}} = K_1,$
$\forall i = k,..,s+k-1$

Multiply both sides of the inequality by '$num_{i,x+1}$'.

$$p_{i,x+1} \times num_{i,x+1} \geq K_1 \times num_{i,x+1}. \tag{7.16}$$

Take the cumulative summation for i = k,…, s+k-1,

$$\sum_{i=k}^{k+s-1} p_{i,x+1} \times num_{i,x+1} \geq \sum_{i=k}^{k+s-1} K_1 \times num_{i,x+1} = K_1 \sum_{i=k}^{k+s-1} num_{i,x+1}. \tag{7.17}$$

Using Equation (7.14) and (7.17), and assumption over the performances, '$p_{i,x+1}$' for i = k,…, k+s-1, we can write,

$$\sum_{i=k}^{k+s-1} p_{i,x+1} \times num_{i,x+1} \geq \\ p_{non-preserved} \times num_{non-preserved} + M \times num_{k,x}. \tag{7.18}$$

Add same term to both sides of (7.18),

$$\sum_{i=k}^{k+s-1} p_{i,x+1} \times num_{i,x+1} + p_{preserved,x} \times num_{preserved,x} \geq \\ p_{non-preserved} \times num_{non-preserved} + M \times num_{k,x} + \\ p_{preserved,x} \times num_{preserved,x}. \tag{7.19}$$

We also know that, $p_{k,x} < M$, using this information in (7.19), we can write,

$$\sum_{i=k}^{k+s-1} p_{i,x+1} \times num_{i,x+1} + p_{preserved,x} \times num_{preserved,x} > \\ p_{non-preserved} \times num_{non-preserved} + p_{k,x} \times num_{k,x} + \\ p_{preserved,x} \times num_{preserved,x}. \tag{7.20}$$

Dividing both sides of (7.20) by 'N_T', and using Equations (7.12) and (7.15) we reach to (7.4) meaning that improvement is observed from level 'x' to level 'x+1'.

b) Suppose, at least one of the new agents have a performance value, $p_{i,x+1} < \frac{p_{non-preserved} \times num_{non\_preserved} + M \times num_{k,x}}{num_{non-preserved,x} + num_{k,x}}$, for some $i = k,..,k+s-1$.

Without loss of generality, we assume that there is only one bad performance, say '$p_{k+s-1,x+1}$'. Choose a suitable 's' and go to **Main Step**, and apply the algorithm this time to new unsuccessful region '$r_{k+s-1,x+1}$' as previously applied to '$r_{k,x}$'.

Remarks: If there is more than one unsuccessful region, the previous algorithm can be used repeatedly for each unsuccessful region. The essence of the proof in Case 3 depends on the following facts: The regions associated with unsuccessful agents are divided into sub-regions. Most of the sub-regions have smaller number of data points compared to initial regions. Hence, training NNs as local agents associated with these sub-regions has more chances of having sufficiently more performance. However, a few sub-regions may contain more data points compared to regions associated with unsuccessful agents since regions associated with successful agents are not preserved and some date points can be stolen from them. For this situation, the local agent development process may fail due to increased number of data points. Hence, for such sub-regions the separation process into new sub-regions is reapplied. The worst situation for the algorithm is observed when all the data points are affected from the separation process at consecutive levels. In such a situation, the algorithm may produce a single different agent for each data point. However, it is always possible to design a very simple NN structure (having a single hidden layer neuron) for a single data point that perfectly memorizes its corresponding target (shown in Section 7.1.5).

7.1.5. Memorizing a Target

Assume that we have a single data point $x^* = (x_1\ x_2\ x_3 \ldots x_m)$ and an associated target value for the data point 't^*'. The NN structure for training this data point possesses a single hidden layer with a single neuron in this hidden layer. The output of NN will be obtained as,

$$(A_n f)(x_1,....,x_m) = \sum_{i=1}^{1} w_i \varphi\left(\sum_{j=1}^{m} a_{ij} x_j + b_i\right). \qquad (7.21)$$

We can write Equation (7.21) in another form as

$$(A_n f)(x_1,....,x_m) = w_1 \times \varphi(a_{11}x_1 + a_{12}x_2 + ... + a_{1m}x_m + b_1). \qquad (7.22)$$

Assume $a_{1j} = 0$ for j = 2,.., m, $w_1 = 1$ and $b_1 = 0$; then we obtain,

$$(A_n f)(x_1,....,x_m) = \varphi(a_{11}x_1). \tag{7.23}$$

Define 'a_{11}' as,

$$a_{11} = \frac{\varphi^{-1}(t^*)}{x_1}. \tag{7.24}$$

Then we obtain,

$$(A_n f)(x_1,....,x_m) = t^*. \tag{7.25}$$

So it is always possible to design a NN having a single hidden layer with one neuron to train a single data point such that the output of the NN is exactly equal to the target of the data point.

7.1.6. Over-Fitting and Under-Fitting:

In the proofs of different cases, the validation data is not considered. However for practical applications validation data is an important point of concern. It is important to find the optimal number of hidden layer neurons such that over-fitting or under-fitting is prevented. Hence, consequently, the NN does not memorize all the training data perfectly and besides it has a good generalization property. Memorizing all the data may result in a poor performance for validation data. To prevent over-fitting (memorizing all the training data), different precautions can be taken:

- The training process of NNs is carried on concurrently with the testing process and the training is stopped when the testing performance starts decreasing.
- Different number of hidden layer neurons can be selected to train the NNs at the training step and than the best NN that demonstrates relatively good performance both in the training and validation can be selected as the agent structure.

Through these strategies, the second one is employed for the practical application in Chapter 5. It is observed that using the second strategy the generalization capacity of the NNs is increased and the ability to classify the validation data properly is maintained.

Another treat for training process of a NN is under-fitting. If the NN has inadequate number of variables, the NN may show poor performance both in training and validation data. Because of the under-fitting and over-fitting treats, the number of hidden layer neurons (which determines the number of variables a NN has) should be chosen such that they are not so scarce to spoil the performance in training and they are not so numerous to spoil the validation performance. Regretfully, given a fixed number of data there is no guarantee that the given NN will not over-fit or under-fit the data since the distribution of targets may be very diverse.

Ideally, if there is a NN whose number of variables (weights and biases) is greater than or equal to the number of input-target pairs, the NN will have a tendency to memorize the data. So it is better to choose a NN structure whose number of variables is smaller than the number of input-target pairs. As a convention 5 to 10 times the difference between the number of input-target pair and number of variables are suitable choices for noise- free data training.

7.2. Theoretical Study 2: Hierarchical Decision-Making Model

7.2.1. The Model

x_i: Input (i = 1,….., n so there are n data).

t_i: the output value that should be obtained (target) for 'x_i'. $t_i \in T = \{o_1, o_2, ..., o_m\}$ such that 'T' is the set of all possible discrete target values and 'o_m' and 'o_1' are the maximum and minimum values a target can be assigned, respectively.

(x_i, t_i): input-target pair (a single data)

$y_{i,k}$: Output (decision) for input 'x_i' at level 'k'.

$y_{i,k-1}$: Output for input 'x_i' at level 'k-1'.

y_{i,drb_k}: Output of the newly developing rule-base at level 'k' for input 'x_i'.

$C_{i,k-1}$: Confidence value of output '$y_{i,k-1}$' for input 'x_i' at level 'k-1' where $0 \le C_{i,k-1} \le 1$.

Define output regions corresponding to different discrete targets as

$$\theta_1 = \left[o_1, \frac{o_1 + o_2}{2} \right), \quad \theta_2 = \left[\frac{o_1 + o_2}{2}, \frac{o_2 + o_3}{2} \right), \quad .., \quad \theta_{m-1} = \left[\frac{o_{m-2} + o_{m-1}}{2}, \frac{o_{m-1} + o_m}{2} \right),$$

$$\theta_m = \left[\frac{o_{m-1} + o_m}{2}, o_m \right].$$

For θ_1, assign $C_{i,k-1} = 1 - \frac{\sum_{j, y_{j,k-1} \in \theta_1} \left| y_{j,k-1} - t_j \right|}{\left| o_m - o_1 \right| \times num\_\theta_1}$ where 'num_θ_1' is the number of outputs where $y_{i,k-1} \in \theta_1$. Hence, for 'θ_1', the confidence value at level 'k-1' is a value between 0 and 1 (namely $0 \le C_{i,k-1} \le 1$).

For θ_2 up to θ_m, assign $C_{i,k-1} = 1$.

Using the proposed decision fusion method, '$y_{i,k}$' is written as

$$y_{i,k} = y_{i,k-1} \times C_{i,k-1} + y_{i,drb\_k} \times (1 - C_{i,k-1}). \tag{7.26}$$

The cost at level 'k-1' is equal to

$$Cost_{k-1} = \sum_{i=1}^{n} \left| y_{i,k-1} - t_i \right|. \tag{7.27}$$

In the same way the cost at level 'k' is equal to

$$Cost_k = \sum_{i=1}^{n} \left| y_{i,k} - t_i \right|. \tag{7.28}$$

Putting Equation (7.26) inside Equation (7.28) we obtain,

$$Cost_k = \sum_{i=1}^{n} \left| y_{i,k-1} \times C_{i,k-1} + y_{i,drb\_k} \times (1 - C_{i,k-1}) - t_i \right|. \quad (7.29)$$

In order to have improvement from one level to the other level, (7.30) should hold

$$Cost_{k-1} - Cost_k > 0. \quad (7.30)$$

If $y_{i,k-1} \in \theta_j, j = 2,..,m$, $y_{i,k} = y_{i,k-1}$ since $C_{i,k-1} = 1$ for θ_2 up to θ_m. Hence Equation (7.29) can be reorganized as

$$Cost_k = \sum_{i=1, y_{i,k-1} \in \theta_1}^{num\_\theta_1} \left| y_{i,k-1} \times C_{i,k-1} + y_{i,drb\_k} \times (1 - C_{i,k-1}) - t_i \right| + \sum_{i=1, y_{i,k-1} \notin \theta_1}^{n-num\_\theta_1} \left| y_{i,k-1} - t_i \right|. \quad (7.31)$$

Similarly, '$Cost_{k-1}$' can be written as addition of two summation terms as

$$Cost_{k-1} = \sum_{i=1, y_{i,k-1} \in \theta_1}^{num\_\theta_1} \left| y_{i,k-1} - t_i \right| + \sum_{i=1, y_{i,k-1} \notin \theta_1}^{n-num\_\theta_1} \left| y_{i,k-1} - t_i \right|. \quad (7.32)$$

Equation (7.33) is obtained by eliminating the common terms in Equations (7.31) and (7.32)

$$Cost_{k-1} - Cost_k = \sum_{i=1, y_{i,k-1} \in \theta_1}^{num\_\theta_1} \left| y_{i,k-1} - t_i \right| - \sum_{i=1, y_{i,k-1} \in \theta_1}^{num\_\theta_1} \left| y_{i,k-1} \times C_{i,k-1} + y_{i,drb\_k} \times (1 - C_{i,k-1}) - t_i \right|. \quad (7.33)$$

Assumption 1: $y_{i,drb\_k} = y_{i,k-1}$ for each output except for a single output (say for i = num_θ_1) where $\forall y_{i,k-1} \in \theta_1$.

Reorganizing Equation (7.33) we obtain,

$$\begin{aligned} Cost_{k-1} - Cost_k = & \sum_{i=1, y_{i,k-1} \in \theta_1}^{num\_\theta_1 - 1} \left| y_{i,k-1} - t_i \right| - \\ & \sum_{i=1, y_{i,k-1} \in \theta_1}^{num\_\theta_1 - 1} \left| y_{i,k-1} \times C_{num\_\theta_1,k-1} + y_{i,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) - t_i \right| + \\ & \left| y_{num\_\theta_1,k-1} - t_{num\_\theta_1} \right| - \\ & \left| y_{num\_\theta_1,k-1} \times C_{num\_\theta_1,k-1} + y_{num\_\theta_1,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) - t_{num\_\theta_1} \right|. \end{aligned} \tag{7.34}$$

Using **Assumption 1** with Equation (7.26), one can achieve $y_{i,k} = y_{i,k-1}$ except for a single output (say $y_{num\_\theta_1,k} \neq y_{num\_\theta_1,k-1}$) for $\forall y_{i,k-1} \in \theta_1$. Hence the two summation terms at Equation (7.34) become equal and they cancel each other and Equation (7.35) is obtained

$$\begin{aligned} & Cost_{k-1} - Cost_k = \left| y_{num\_\theta_1,k-1} - t_{num\_\theta_1} \right| - \left| y_{num\_\theta_1,k} - t_{num\_\theta_1} \right| = \\ & \left| y_{num\_\theta_1,k-1} - t_{num\_\theta_1} \right| - \left| y_{num\_\theta_1,k-1} \times C_{num\_\theta_1,k-1} + y_{num\_\theta_1,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) - t_{num\_\theta_1} \right|. \end{aligned} \tag{7.35}$$

Assumption 2: $\left| y_{num\_\theta_1,k-1} - t_{num\_\theta_1} \right| - \left| y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1} \right| > 0.$

By adding and subtracting the same term '$C_{num\_\theta_1,k-1} \times t_{num\_\theta_1}$' inside the absolute value term, one can simply write,

$$\begin{aligned} & \left| y_{num\_\theta_1,k-1} \times C_{num\_\theta_1,k-1} + y_{num\_\theta_1,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) - t_{num\_\theta_1} \right| = \\ & \left| \begin{aligned} & y_{num\_\theta_1,k-1} \times C_{num\_\theta_1,k-1} + y_{num\_\theta_1,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) \\ & - t_{num\_\theta_1} + C_{num\_\theta_1,k-1} \times t_{num\_\theta_1} - C_{num\_\theta_1,k-1} \times t_{num\_\theta_1} \end{aligned} \right|. \end{aligned} \tag{7.36}$$

Taking the common terms inside the same parenthesis we obtain,

$$\begin{aligned} & \left| y_{num\_\theta_1,k-1} \times C_{num\_\theta_1,k-1} + y_{num\_\theta_1,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) - t_{num\_\theta_1} \right| = \\ & \left| (y_{num\_\theta_1,k-1} - t_{num\_\theta_1}) C_{num\_\theta_1,k-1} + (y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1}) \times (1 - C_{num\_\theta_1,k-1}) \right|. \end{aligned} \tag{7.37}$$

Using the property, $|a|+|b| \geq |a+b|$, one can easily achieve

$$\begin{aligned}&\left|(y_{num\_\theta_1,k-1} - t_{num\_\theta_1}) \times C_{num\_\theta_1,k-1} + (y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1}) \times (1 - C_{num\_\theta_1,k-1})\right| \leq \\ &\left|(y_{num\_\theta_1,k-1} - t_{num\_\theta_1}) \times C_{num\_\theta_1,k-1}\right| + \left|(y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1}) \times (1 - C_{num\_\theta_1,k-1})\right|.\end{aligned} \tag{7.38}$$

Knowing that $0 \leq C_{i,k-1} \leq 1$ for all 'i' values, the result is

$$\begin{aligned}&\left|(y_{num\_\theta_1,k-1} - t_{num\_\theta_1}) \times C_{num\_\theta_1,k-1}\right| + \left|(y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1}) \times (1 - C_{num\_\theta_1,k-1})\right| = \\ &\left|y_{num\_\theta_1,k-1} - t_{num\_\theta_1}\right| \times C_{num\_\theta_1,k-1} + \left|y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1}\right| \times (1 - C_{num\_\theta_1,k-1}).\end{aligned} \tag{7.39}$$

Assumption 2 now implies,

$$\begin{aligned}&\left|y_{num\_\theta_1,k-1} - t_{num\_\theta_1}\right| \times C_{num\_\theta_1,k-1} + \left|y_{num\_\theta_1,drb\_k} - t_{num\_\theta_1}\right| \times (1 - C_{num\_\theta_1,k-1}) < \\ &\left|y_{num\_\theta_1,k-1} - t_{num\_\theta_1}\right| \times C_{num\_\theta_1,k-1} + \\ &\left|y_{num\_\theta_1,k-1} - t_{num\_\theta_1}\right| \times (1 - C_{num\_\theta_1,k-1}) = \left|y_{num\_\theta_1,k-1} - t_{num\_\theta_1}\right|.\end{aligned} \tag{7.40}$$

Using (7.40), (7.38) and Equations (7.39), (7.37), and (7.36) with **Assumption 2**, we obtain

$$\begin{aligned}&\left|y_{num\_\theta_1,k-1} - t_{num\_\theta_1}\right| - \\ &\left|y_{num\_\theta_1,k-1} \times C_{num\_\theta_1,k-1} + y_{num\_\theta_1,drb\_k} \times (1 - C_{num\_\theta_1,k-1}) - t_{num\_\theta_1}\right| > 0.\end{aligned} \tag{7.41}$$

Consequently, $Cost_{k-1} - Cost_k > 0$, which means (7.30) is satisfied.

CHAPTER 8

CASE STUDY 2: LOTTERY DATA ANALYSIS (CS2)

The prediction of amount of sales of units of some product provided that a certain amount of a priori information about previous sales and price of the product is available is a common problem in economics. Particularly we are interested in the lotto data analysis. Sales of lottery tickets depend on many economical issues. The present and past prices of the tickets, the present and past prizes of the lottery and the number of previous sales of tickets all influence the ticket sales at present drawing. Such a problem was investigated in [52]. In this study Genetic Programming (GP) is used in order to construct different models that predict the number of future lottery ticket sales in Israel lottery. Indeed this question can also be presented as time series data mining problem and many different time series data mining methods can be used in order to predict the amount of sales.

8.1. Problem Definition

In [52], in one of the developed models using GP (only using the present and past values of the prizes and the present and the past values of the price of lottery ticket to calculate the amount of ticket sales at the present drawing), it is emphasized that the best function found at the end of GP search is,

$$A_0 = \frac{B_0 + 2C_0}{C_3^2 + C_3 \times (3 + C_8^2) + \dfrac{B_7}{0.137B_7 + C_3 \times (-2.8 + 5.8B_1 + B_2 + C_2 + 2C_8)}}. \quad (8.1)$$

In Equation (8.1), ‘A’ represents the number of sales ‘B’ represents the prize of the lottery and ‘C’ represents the price of the tickets. The numerical suffixes refer to

lags. For example, 'C_2' represents second lag of 'C' or 'C(t-2)', namely the price of the tickets of two previous drawing.

The real data associated with this problem was not available. It has been decided to create the relevant data artificially, based on the GP model in equation (8.1). Equation (8.1) gives the number of ticket sales (i.e., A) at the present drawing. Unfortunately 'A' value at present time depends on 'B' and 'C' values of present and past drawings and 'B' and 'C' values are not available, either. For this reason, they have been generated artificially. It is assumed that the lottery is drawn every week in a year and the price of the tickets is increased linearly every week due to the announced (predicted) inflation rate in the year (which has been taken as 5%). Initially a base price is determined for each year. Then, using this base price the ticket price of each week is calculated. For example, if the base price of a ticket is 'K' at the first week of the year; at the second week of the year, at the third week of the year and at the last week of the year it will be equal to $K + \left(\frac{1}{52} \times \frac{5}{100} \times K\right)$ $K + \left(\frac{2}{52} \times \frac{5}{100} \times K\right)$ and $K + \left(\frac{51}{52} \times \frac{5}{100} \times K\right)$, respectively. Finally at the first week of the next year it will be $M = K + \left(\frac{5}{100} \times K\right)$ which is the base price of the next years' tickets. The first week price of the ticket is, then taken as the base price 'M'. For each year, the same procedure is repeated to determine the ticket price for forthcoming weeks. The procedure is carried for 1050 weeks and price of a ticket for each week in this duration is obtained. In Fig 8.1, the price of a ticket at each week is shown.

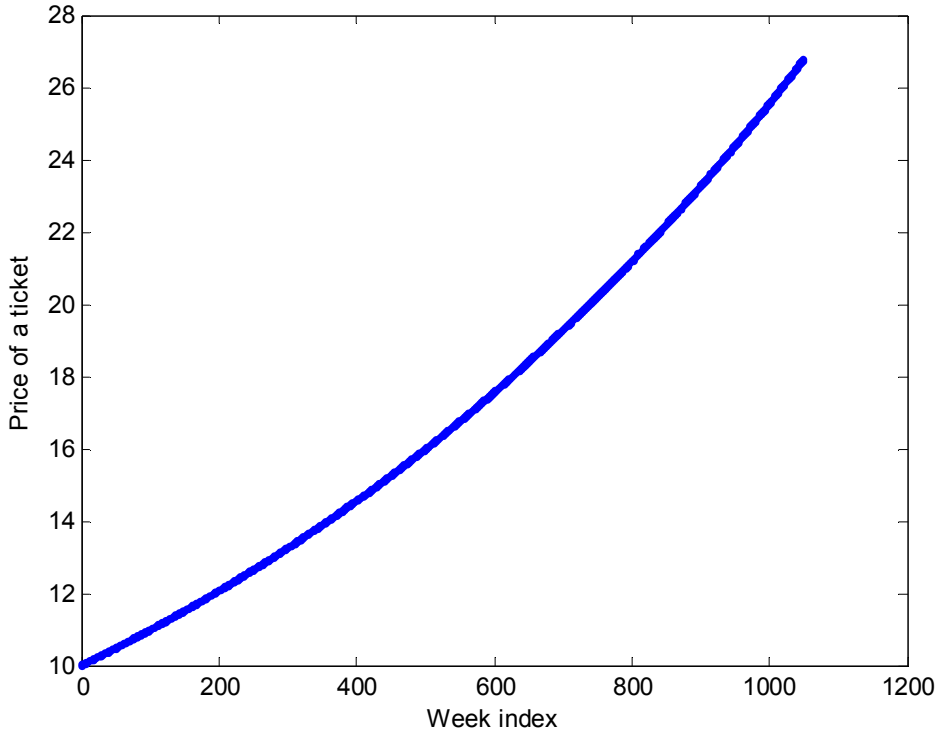

Fig. 8.1: **The price of a ticket per week.**

In order to generate the prize values a different procedure is applied. A base price is set at the first week of each year. In a week of a year if lottery is won by somebody, the next week's prize is announced as the base price determined for the year. Otherwise (if current week's lottery was not won by anybody), the lottery prize for the next week is increased by 50 % of the current week's lottery prize. It is assumed that, at the end of sixth consecutive draws at most (in a duration of 6 weeks) somebody is sure to won the lottery and also that the draw is a random process such that the probability that it will be won in the current week is $\frac{1}{6-n}$ where 'n' is the number of weeks passed after the lottery was last won. The variation of the prize of the lottery for each week is shown in Fig. 8.2.

Equation (8.1) is used to generate the number of tickets sold in a week. However the previous and present values of the prize and price are necessary to evaluate the number of tickets sold at the present drawing and the maximum lag amount in

Equation (8.1) is 8 units; 9 consecutive values of the price of a ticket and the price are necessary to evaluate the number of tickets sold in a week. For this reason initial 9 values of the number of ticket sales are evaluated randomly (first nine weeks of first year) than remaining values are calculated using Equation (8.1). The number of tickets sold per week is shown in Fig. 8.3.

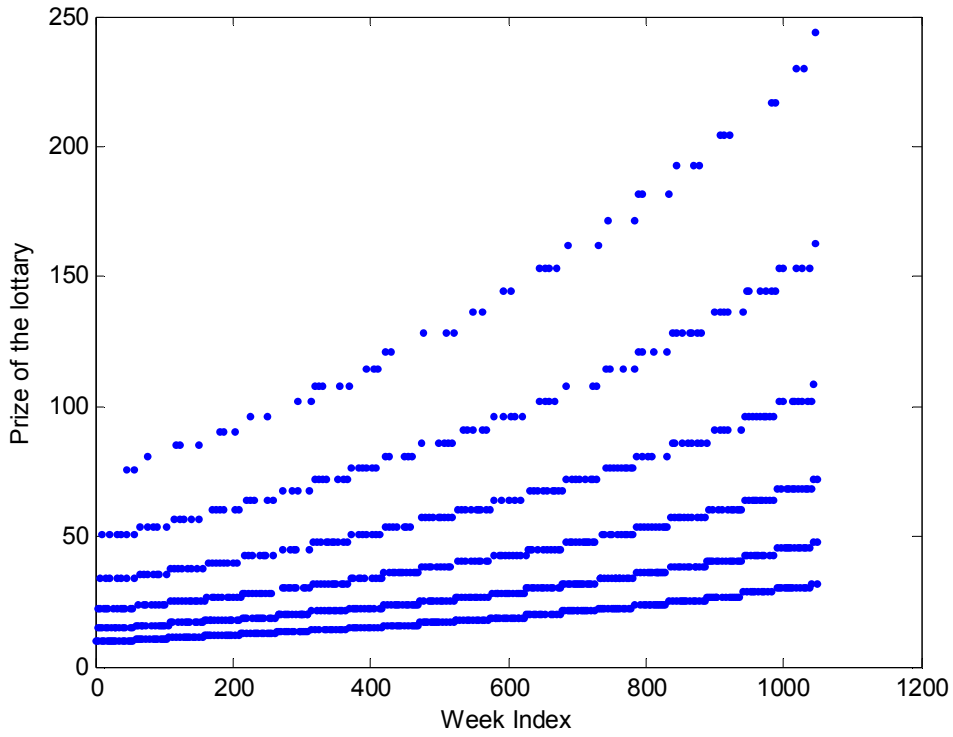

Fig. 8.2: The prize of the lottery at each week.

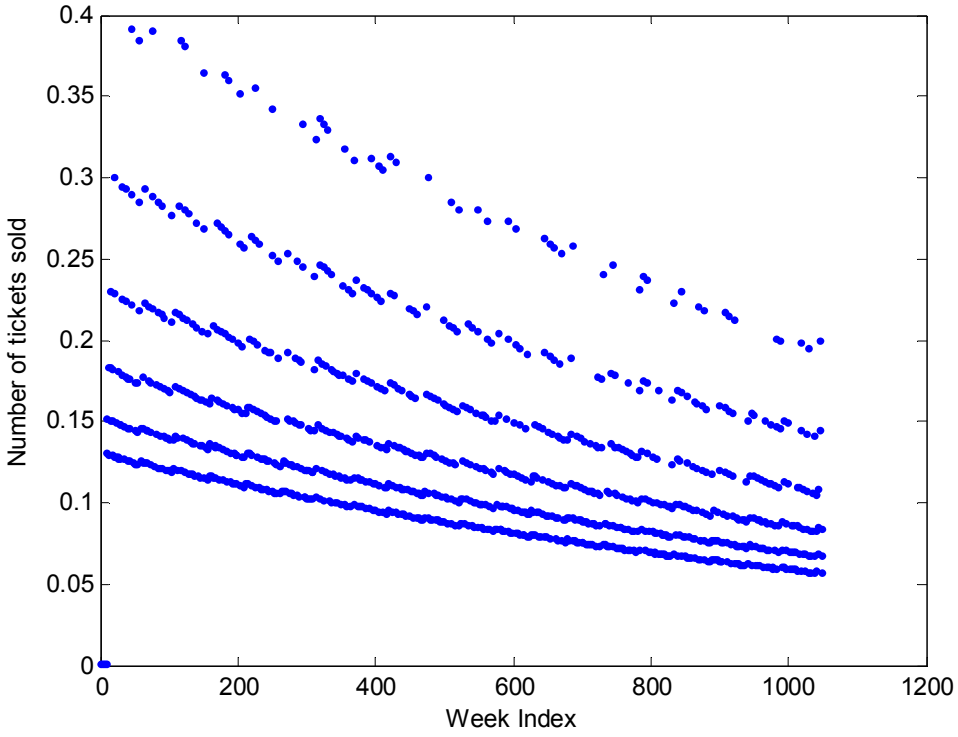

Fig. 8.3: **The number of tickets sold at each week.**

The data created for 3 different characteristics of lottery data (price of a ticket, prize of lottery and number of tickets sold) are used as input-target pairs of the multi-level DM models to be developed. However it should be transformed into a new structure before being used. For this reason the available data has been pre-processed. The three characteristics of lottery data are normalized and all the values are brought into a region where maximum value is 1. Since all the data values are positive, the minimum of all the data values is also greater than 0.

The aim is to develop a multi-level DM model that predicts (decides on) the amount of tickets sold (in normalized units) when the normalized price of the tickets and the normalized prize of the lottery for the present and past weeks are available. For this job, the proposed multi-level DM models are used.

8.2. Application 1: HDM Model

In the first application HDM model is used. A single agent is developed at the first level that predicts the number of tickets sold per week. The agent is a NN that utilizes the past and previous price of tickets and the past and present prize of the lottery as inputs and the corresponding present number of tickets sold as the target. After the agent is developed, a performance function is suggested for the agent. Using the performance function the predictions of the first level agent is brought into the second level. The second level agent is made up of two parts: A static part that is composed of the first level decisions of the first level agent and the performance function associated with the first level decisions and a dynamic part which is a new NN structure. The decision of the dynamic part and the decision of the static part are fused as proposed in Chapter 3, using a convex fusion equation. The resulting fused decisions are used in order to train the NN in second level.

8.2.1. The First Level

In the first level the agent structure is a feed-forward NN. Its inputs are the price of the tickets and the prize of the lottery for five consecutive weeks starting from present week down to 4 previous weeks. The output of the network is the prediction of the number of tickets sold at the present week. The number of hidden layer neurons is chosen to be 10; this choice is based of the number of data points that will be included in the training process. Back propagation algorithm is used in order to train the NN. The cost function to train the NN is chosen as,

$$Cost = \sum_{i=10}^{1050} \left(p_{1,i} - t_i \right)^2 . \tag{8.2}$$

In Equation (8.2), 't_i' is the target value for index 'i' (the actual number of tickets sold), '$p_{1,i}$' is the predicted ticket sales at the first level. The index starts from 10 for comparison purposes with the function obtained by GP shown in Equation (8.1).

Back propagation algorithm is applied for 12000 steps and the first level agent is developed. In order to determine the partial success of the agent in different decisions, a performance function associated with the agent is constructed. The first level predictions are divided into 10 equal regions according to their values (a prediction is a value between 0 and 1 and the size of each region is 0.1 units). For each region the absolute total cost is calculated as,

$$Ac_{R_k} = \sum_{i=1, p_{i,1} \in R_k}^{n_k} \left| p_{1,i} - t_i \right|. \tag{8.3}$$

In Equation (8.3), ' Ac_{R_k} ' is the absolute total cost for the region 'R_k', 'n_k' is the number of predictions which are inside the boundaries of region 'R_k'. The performance (performance function) of predictions in region 'R_k' is calculated using,

$$P_{R_k} = 1 - \frac{Ac_{R_k}}{n_k}, \tag{8.4}$$

where,

$$0 \leq P_{R_k} \leq 1. \tag{8.5}$$

The performance function for different decision regions is shown in Fig. 8.4. The confidence value of any prediction is calculated using the performance function.

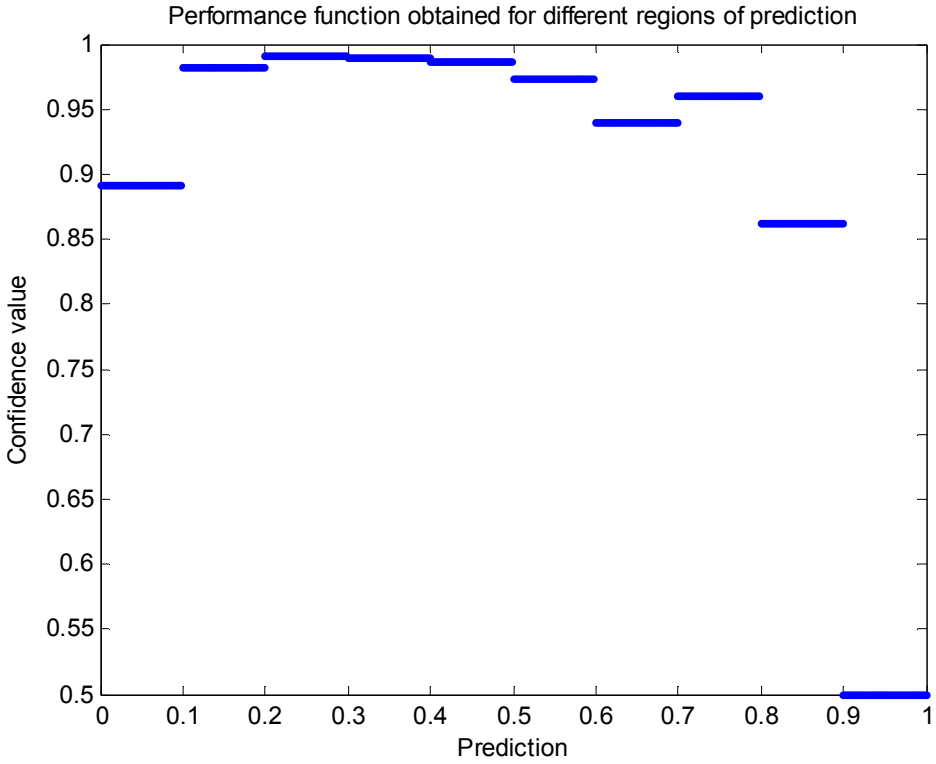

Fig. 8.4: **The performance function defined for the decision regions.**

8.2.2. The Second Level

The predictions obtained in the first level and predictions' confidence value obtained from the performance function are transferred into second level. The decision of a new NN structure (which has the same structure as the first level NN) and the decisions of first level agent are fused using,

$$p_{2,i} = c_{1,i} \times p_{1,i} + (1 - c_{1,i}) \times p_{n\_net}. \tag{8.6}$$

In Equation (8.6), '$p_{2,i}$' is the second level prediction, '$c_{1,i}$' is the confidence value of first level prediction obtained from the performance function, '$p_{1,i}$' is the first level prediction and 'p_{n_net}' is the prediction of the new NN in the second level. In the second level the cost function is,

$$Cost = \sum_{i=10}^{1050} \left(p_{2,i} - t_i \right)^2. \tag{8.7}$$

Using the back propagation algorithm the NN is trained. However, one should be careful while evaluating the gradient of the cost with respect to the parameters of the NN since cost depends on '$p_{2,i}$' and '$p_{2,i}$' depends on 'p_{n_net}'. So using the chain rule the partial derivative of the cost function with respect to NN parameters can be expressed as,

$$\frac{dCost}{dparameter_k} = \frac{dCost}{dp_{2,i}} \times \frac{dp_{2,i}}{dp_{n\_net}} x \frac{dp_{n\_net}}{dparameter_k}. \tag{8.8}$$

The NN is trained for 12000 iterations and the second level agent is obtained which determines the second level prediction. The total absolute cost of an agent is calculated using,

$$Abs\_Cost = \sum_{i=10}^{1050} \left| p_{k,i} - t_i \right|, \tag{8.9}$$

where, 'Abs_Cost' is the total absolute cost and '$p_{k,i}$' is the prediction for of k^{th} level agent. The performance of the agent is calculated using,

$$Per = 1 - \frac{Abs\_Cost}{1041}. \tag{8.10}$$

The total absolute cost and corresponding performance values for the first level and the second level agents are given in the Table 8.1.

Table 8.1: **The total absolute cost and performance values of the first and the second level agents and the GP model.**

	Total Absolute Cost	Performance
The first level	17.4843	0.9832
The second level	14.2808	0.9863
GP model	0	1

The performance and total absolute cost value obtained using GP model is sure to be equal to 0 and 1, respectively, since the targets (the ticket sales at each week) are obtained using (8.1). The change of total absolute cost for increasing number of training iterations to develop agents of the first level and the second level are shown in the Figs 8.5 and 8.6 respectively.

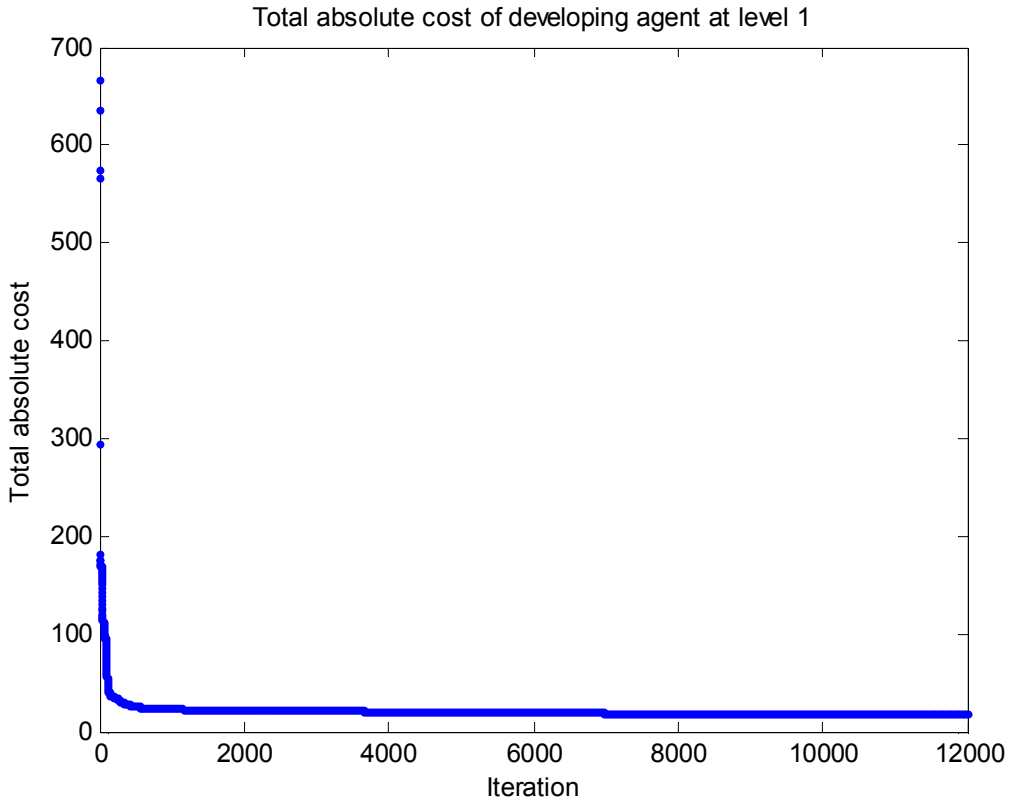

Fig. 8.5: **The total absolute cost of developing agent at the first level.**

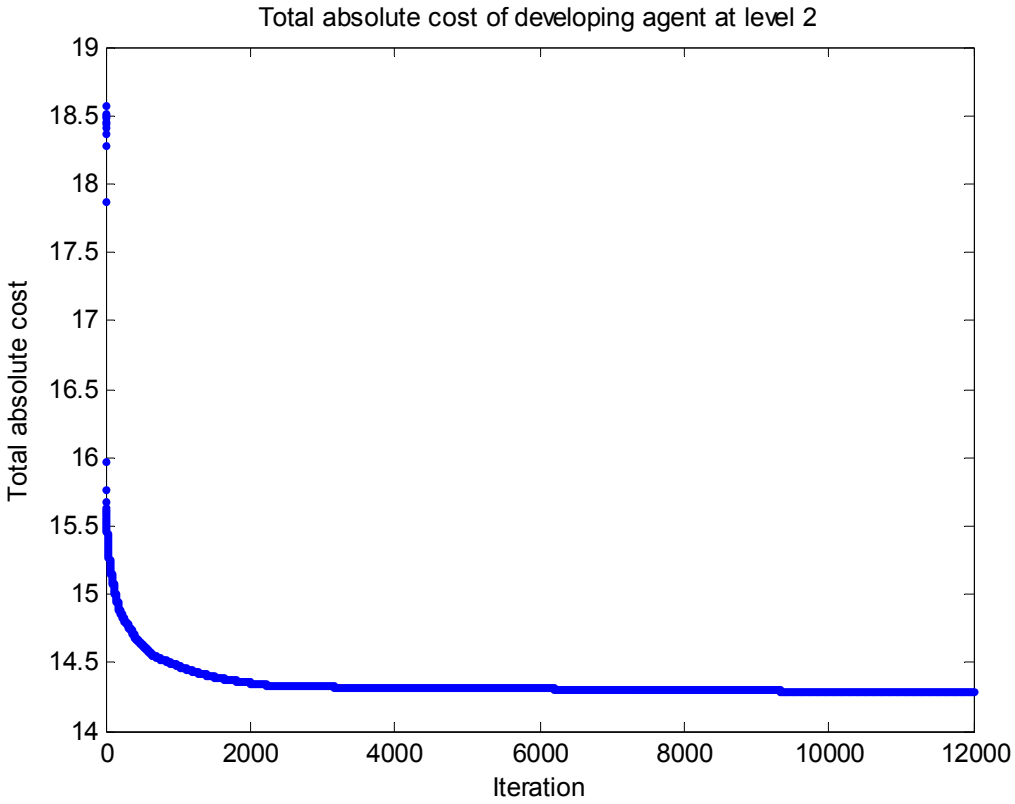

Fig. 8.6: **The total absolute cost developing agent at the second level.**

The total absolute cost value of the second level agent at the 10^{th} iteration step is smaller than the total absolute cost value of the first level agent obtained at the final iteration step (iteration step 12000). Using the HDM model, the total absolute cost of the second level agent is decreased much more with respect to the first level agent. The result is an indicator of the strength of HDM model.

8.2.3. The Effect of Noise

Noise (uniform noise) is also added to the input data and the effect of noise is investigated both for the agents obtained at HDM model and the function obtained by the GP model using Equation (8.1). The SNR for the noisy data is 42.7688. The performance and total absolute cost values obtained at the noisy environment for

the first level agent, for the second level agent, and for the GP function in Equation (8.1) are given in Table 8.2.

Table 8.2: **The total absolute cost and the performance values obtained for the first agent, for the second level agent and for the GP model.**

Agents at noisy data	Performance	Total Absolute Cost
The first level	0.9707	30.4958
The second level	0.9734	27.6655
GP function	0.9846	15.9979

As seen from the results (Table 8.1 and Table 8.2), HDM model demonstrates increasing performance both for the training and the testing data (in noisy environment) as levels are increased. Although the data is obtained using the GP model, HDM model exhibits comparable performance values with the GP model.

8.3. Application 2: LDM-ML Model

8.3.1. The First Level

The second application is the development of LDM-ML model for the DM problem. LDM-ML model depends on clustering the input data. But before clustering, the input data is rearranged: The inputs to be used in the LDM-ML model are composed of prize of lottery and the price of lottery ticket for five consecutive weeks (from week ‘t’ down to week ‘t-4’) and the corresponding target is the number of ticket sales (at week ‘t’). 1041 input and their corresponding targets are formed. The inputs are clustered into 2 groups using the k-means algorithm. If an input is closest to a specific cluster center, it belongs to that cluster. A single NN is trained for each cluster. The NNs have 4 hidden layer neurons and a single output neuron. The output of the NNs is the prediction for the

corresponding input. In order to make the NNs concentrate directly on the cluster centers, the cost function is chosen as

$$Cost_k = \sum_{i=1}^{n_k} e^{-\frac{\|x_i - c_k\|^2}{\sigma^2}} (p_i - t_i)^2, \tag{8.11}$$

where, '$Cost_k$' is the training cost function of the NN developing for the cluster 'k', 'n_k' is the number of input in cluster 'k', 'x_i' is an input in cluster 'k', 'c_k' is the cluster center of cluster 'k', 'p_i' is the prediction for input 'x_i' and 't_i' is the corresponding target for input 'x_i'. The term 'σ' makes the training concentrate on the cluster center. In the training simulations, 'σ' is taken as 0.5. The NNs are trained for 12000 iterations. The local performance of each trained NN is calculated using,

$$Per_k = 1 - \frac{\sum_{i=1}^{n_k} |p_i - t_i|}{n_k}, \tag{8.12}$$

where, 'Per_k' is the local performance of the NN developed for cluster 'k'. After local decisions are performed the decision fusion technique suggested in Chapter 5 is used and the final decision in the first level are obtained (the local agent developed for the cluster determines the decision for all the data inside the cluster). The total performance in any level is then equal to,

$$Tot\_per = \frac{\sum_{i=1}^{num\_cluster} Per_k \times n_k}{\sum_{i=1}^{2} n_k}, \tag{8.13}$$

where, 'Tot_per' is the total performance at the corresponding level and 'num_cluster' is the number of clusters (regions) available at that level. The local performance values for each local agent for the first level in their local regions and

number of data points included by the corresponding regions are shown in Table 8.3.

Table 8.3: **The local performance of the first level agents and the number of data enclosed by the local regions.**

The first level	Local Performance	Number of input data points inside the region
$Agent_{1,1}$ (The first level)	0.9259	417
$Agent_{2,1}$ (The first level)	0.9718	624

The total performance in the first level is calculated as 0.9534.

8.3.2. The Second Level

As seen from Table 8.3, the local performance of '$Agent_{1,1}$' is worse than the local performance of '$Agent_{2,1}$'. In the second level '$Agent_{2,1}$' is retained in the local agent set. However '$Agent_{1,1}$' is deleted from the agent set and the input data covered by '$Agent_{1,1}$' is re-clustered using k-means algorithm. Resulting new clusters and cluster centers are utilized in the training process. However, due to the new clusters some part of the input data belonging to '$Agent_{2,1}$' passes to these new clusters. In the second level, the number of input data points each cluster possesses is shown in Table 8.4.

Table 8.4: **The number of input possessed by the clusters in the second level.**

The second level	$Agent_{1,2}$	$Agent_{2,2}$	$Agent_{3,2}$=$Agent_{2,1}$
Number of input	178	302	561

'$Agent_{1,2}$' and '$Agent_{2,2}$' are new agents and they are trained using Equation (8.11) using the principles in the first level (the same NN structure and the same training technique). '$Agent_{2,1}$' (or with its new name '$Agent_{3,2}$') is directly copied from the first level to second level. The number of input data points handled by '$Agent_{3,2}$' is decreased. However due to the structure of the chosen Cost function, a drastic decrease in its new local performance value is unexpected. Table 8.5 shows the local performance values of the second level agents in their local regions.

Table 8.5: **Local performance of the second level agents.**

The second level	$Agent_{1,2}$	$Agent_{2,2}$	$Agent_{3,2}$
Local performance	0.9871	0.9956	0.9717

The total performance in the second level is calculated as in the first level and it is found as 0.9813. As seen from the results, at the training step, an incremental change in the total performance value from the first level to the second level has been achieved.

8.3.3. The Effect of Noise

When noise is added (whose distribution and SNR is given in Section 8.2.3), it is still observed that there is an increase in the total performance values. The results

are tabulated in Table 8.6 for the first level and Table 8.7 for the second level, respectively.

Table 8.6: The local performances of the agents for the first level (with noise).

The first level	$Agent_{1,1}$	$Agent_{2,1}$
Local performance (at noisy environment)	0.9263	0.9681
Number of input data possessed	421	620

Table 8.7: The local performance of the agents for the second level (with noise).

The second level	$Agent_{1,2}$	$Agent_{2,2}$	$Agent_{2,3}$
Local performance (at noisy environment)	0.9832	0.9877	0.9677
Number of input data possessed	179	302	560

The total performance for the noisy data at the first level and the second level are 0.9512 and 0.9762, respectively. This shows that the total performance also increases for the test data from the first level to the second level with LDM-ML model.

CHAPTER 9

CONCLUSIONS

9.1. The Fundamental Content of the Dissertation

This dissertation is devoted to develop new multi-level DM models. Two multi-level DM models are designed and their performances are exhibited over two case study problems. As a matter of fact, CS1 is a fault detection problem for a dynamical system. The dynamics of the system are defined by a set of non-linear differential equations. Some artificial faults are generated by varying several system parameters. The main aim of the fault detection process is to determine which parameter is faulty and how much faulty it is. Using the fault-free dynamics and faulty dynamics of the system an error data related with the variables of the system is constituted associated with different fault conditions (i.e., multiple or single fault scenarios). The error data and the corresponding fault characteristics are used as the input-output pairs respectively to develop the multi-level DM models. As the multi-level DM models are developed, their efficiency for the validation data and consistency in noisy environments are tested in different applications. The multi-level DM models are also compared with some well-known classification methods.

CS2 is about lotto data analysis. The multi-level DM models are also applied to this problem with success.

There are also two theoretical studies to illustrate the efficiency of the structure of the proposed multi-level DM models. In these theoretical studies, the models are investigated mathematically from the point of view of convergence towards some desired performance. The improvement on some cumulative performance measure

accounting for all the decisions at successive levels are revealed and proven based on certain a priori assumptions.

9.2. The Strong and Weak Aspects of Multi-Level Models

9.2.1. Training and Validation Scenarios

For CS1, both of the employed models show good performance in the noise-free training and validation scenarios. The performance measures mostly tend to improve in all of the applications. While passing from a lower level to one higher level, the cumulative performance measures related with all the decisions usually improve. The application using LDM-ML model (model used in Chapter 5) has better performance values compared to the applications using the HDM model (i.e., the applications of Chapter 3 and Chapter 4). Besides LDM-ML model has better performance values compared to the Bayes optimal classifier and the Adaboost (i.e., Chapter 6, Section 6.1) classification methods, as well. Even though the Bayes optimal classifier [48] and the Adaboost [27, 28, 49] methods are tuned optimally, they are dominated by LDM-ML model. As the resolution increases (i.e., resolution corresponds to the number of clusters the error data is divided into. In the LDM-ML model, as progressing from a lower level to a higher one, the number of clusters is increased and for each newly created cluster, a new decision maker is developed. Similarly the Bayes optimal classifier uses the same clusters to assign the posteriori and priori probabilities which determine the functionality of the method) the Bayes optimal classifier also exhibits improved performance values. However, Adaboost generally have declined performance values in the same process. There are two reasons for the poor performance of Adaboost. Firstly, the local agents of the LDM-ML model are chosen as the weak classifiers of Adaboost. The fusion of decisions using Adaboost diminishes the effect of local decision makers which function as weak classifiers. Hence, the fusion technique applied in Chapter 5 is more advantageous compared to Adaboost for the local agents developed. Besides, as the number of clusters is increased, the

number of local agents getting inside the fusion operation increases. The local agents in Chapter 5 are developed only to perform consistent decisions inside the cluster and mainly around a focal (center) point chosen inside the cluster. The fusion technique applied in Chapter 5 dictates that the local agent developed for a cluster is the only authority for a data point inside the cluster. However, Adaboost applies a fusion equation similar to weighted averaging which is mixed with a threshold operation. Hence, a local decision maker away from the data has also some effect over the decision. Unfortunately, a local agent away from the data may usually have unsatisfactory decisions, since it is basically developed to operate within its associated cluster. As a result, as the number of clusters increases, the performance of the Adaboost algorithm diminishes. It is possible to develop weak classifiers for Adaboost that work globally (i.e., not restricted to the data inside a specific cluster). This is also performed (i.e., Chapter 6, Section 6.3). The performance is close to first level of first application (i.e., Chapter 6, Section 6.2). However, the required computation time to develop so many classifiers (i.e., in Chapter 6, Section 6.3, the number of developed weak classifiers is 35) is very long, whereas the computation time required to develop local agents is very small. This situation is one of the drawbacks of the Adaboost algorithm. This is because of the fact that the training time for the global weak classifiers increases exponentially.

For CS2, both of the proposed models give satisfactory results both for the training and validation data (validation data is used to test the performance of the models in noisy environments). Both of the models work well and no decrease in performance values from a lower level to higher level is observed.

9.2.2. Noise

For CS1, a group of applications benefiting from the multi-level DM models (i.e., the applications in Chapter 3 and some of the applications in Chapter 4) have poor performance in noisy environments. In the applications in Chapter 3, we have observed that as one progress from a lower level to one higher level, the

performance measures diminish for all applied types of noise (i.e., Gaussian noise and uniform noise) at different SNR settings (i.e., high, medium, and low SNRs).

The applications in Chapter 4 have better noise resistance compared to applications in Chapter 3. If the SNR is high, improvement is observed in the performance measures for both noise types. On the contrary, when the amount of noise is increased, the performance measures from a lower level to one higher level exhibit a mixed character. Sometimes the performance measures improve and sometimes they deteriorate. Indeed, the performance measures generally saturate around some values and fluctuate around the saturated values.

There are two important reasons for observing such different noise endurance characteristics in the applications of Chapter 3 and Chapter 4. Primarily, although the same DM model is used in the applications, the styles in which the models are applied are different. In the applications of Chapter 3, a single agent is developed at each level. The single agent developed in a level helps the development of the single agent in one higher level. However, the applications of Chapter 4 use a multi-agent development idea. The developed agents of a level help the development of the different structured agents of one higher level. This multi-agent development idea provides robustness for the agents developed. The decision fusion equations used in Chapter 4 provides a framework for aggregation of decisions of different decision makers. Secondly, the rule-bases used in Chapter 3 and Chapter 4 are different in structure and they benefit from different data evaluation techniques. In Chapter 3, raw data is directly evaluated in the rule-base. The rule-base utilizes the principles of fuzzy logic to assign an output for the input data. On the contrary, the rule-bases in Chapter 4 are made up of sequences with output attributes. The raw data fed into the rule-base is first of all pre-processed according to the cluster content in the sequences of the rule-base. The pre-processing of data depends on a kind of distance measure between the data and the cluster content in the sequence. This distance measure provides robustness in noise applications. A fluctuation in the position of the data because of applied noise is compensated for due to pre-processing. Instead of directly calculating membership

values using raw data, a distance measure is used to calculate the membership value. There is a second advantage of using the sequences in the rule-bases. Some of the rule-bases include sequences having more than one cluster content inside (i.e., sequence depths are greater than 1). For these sequences, the degree of membership for a sequence about a data for any time instant "t" (i.e., $data_t$) is determined according to the distance measure calculated between the cluster content in the sequence and the limited number of consecutive data corresponding to the time instants up to time instant, 't'. The consecutive data points are usually very close to each other. The membership assignment procedure uses many different reference points (different cluster centers). Due to the increased number of references in the calculation of the membership value, the fluctuations caused by noise over the data usually tend to be less effective on the membership assignment process. Hence the membership value assignments are less effected from noise.

The type of noise has a very limited effect on the performances. Usually similar performances are observed at the same levels for the applications of Chapter 3 and Chapter 4 when noises with different distributions are applied.

In Chapter 5, the effect of noise over the LDM-ML model has limited effect if the SNR is high. This result is valid for both of the noise types with different distributions. Similarly, when progressing from a level to one higher level the performances usually improve both for the training and the validation scenarios.

When the results of the Bayes optimal classifier are compared with the results of the local DM model, the Bayes optimal classifier seem to have better performance for high and medium noise cases (i.e., when SNR is low or medium). However when the noise level is low (i.e., SNR is high), LDM-ML model shows better performance. As the number of clusters is increased, if the amount of noise is low, noisy data generally tends to remain very close to its original spatial position. The neural network developed for a cluster determines a non-linear surface that tries to describe the target of the data inside the cluster. Hence, a small amount of

fluctuation in the data does not change the output assigned for the data point much if a talented neural network developed for the cluster (i.e., a neural network having good generalization ability is used as the agent structure to determine the decisions). What is observed for a high SNR case is the explained above. When the noise amount is increased (i.e., medium or low SNR) the neural network structure starts becoming ineffective for two reasons. First of all, the possibility for data to jump to another cluster increases (i.e., that means the noisy data is processed by another agent developed for another cluster). Secondly, due to the non-linear processing structure of the neural network it is often possible to observe diminished efficiency in DM for a data point when noise amount is medium or high.

However, there is a drawback of the Bayes optimal classifier. When the noise level is high, (i.e., the SNR is low) if the number of clusters are further increased (especially while passing from the third level to fourth level as in Chapter 5) there is a drastic decrease in performance. When the SNR is low, data drifts more from its original position. Besides if the number of clusters is large, it is more probable that the data point will jump to the territory of another cluster. If the data remains in the same cluster, its classification will be indifferent due to the way in which the posteriori and priori probabilities are defined. Otherwise (i.e., if the data passes to another cluster) it is possible that the data will possess a new classification. As a result, increasing the resolution does not guarantee that the Bayes optimal classifier will be more efficient.

For CS2, the effect of noise with high SNR over the performance of the proposed DM models is limited. Both of the models show good performance for applications with noise.

9.2.3. Computation Time

The computation time required to develop the agents is another important issue. Generally, developing agents using GA is a time-consuming process. For this

reason, in Chapter 3 and Chapter 4, the optimization simulations for the applications have longer durations than simulations in Chapter 5. Besides, to develop an agent in an application of Chapter 4 is more demanding and more computation time is required since an additional pre-processing is performed for the raw data.

In Chapter 5, the computation time required to develop all the local agents generally takes less time than the time required to develop an agent in Chapters 3 and 4. There are two reasons for this fact. Primarily, the local agents are developed using a limited amount of data inside the clusters. Secondly, a global optimization algorithm like GA is not used to develop the local agents. GA starts optimization from many different points. However in Chapter 5, a local search algorithm starting from a single point is used. Since the number of data points inside a cluster is limited, a local search algorithm has more potential to yield good results.

9.2.4. Clustering

LDM-ML model depends on clustering the input data. There are many methods of clustering the data. k-means algorithm is one of them. If the clustering algorithm or the parameters of the clustering algorithm are not chosen suitably, it will create some problems. First of all, if the clustering operation is not performed properly, it is possible to observe input data point in the same cluster having very diverse characteristics (for our case, if two or more input data point very close to each other have very different target values, this is an indicator of diverseness). Even if the clustering produces some hardly manageable local regions, it is the duty of the LDM-ML model to compensate the side effects of clustering since LDM-ML model continuously divide the local regions where the local agents show poor performance, into new sub-regions and decrease the possibility of insufficient clustering by this way. The agents of these new local regions are responsible from a reduced number of inputs, so they have more chance to be successful. As the result, clustering type and clustering parameters may sometimes create some

problems. However the problems created due to bad clustering can be overcome by the model.

9.2.5. Applicability of the Models to Different Problems

One of the important issues about the models is their applicability. The models proposed in this thesis can be developed for and applied to any DM problem when we know the circumstances that decisions will be accomplished (the input for whom the decision will be given) and the correct decisions of the corresponding circumstances (the targets for the inputs). The input may be pre-processed (clustering, normalization, bringing the data into a time series data structure) before using in the models to obtain better performance according to the requirements.

The basic principle to apply the DM models to any DM problem is defining a performance measure for the agent or for the decisions that helps to transfer the decisions obtained in a lower level partially or totally to an upper level. Secondly, a suitable agent structure should be selected considering the size of the input data, the problem type and the configuration of the data. If these requirements are satisfied, the DM models can be applied to all kinds of DM problems.

9.3. What are the Additional New Ideas to Improve the Functionality of the Proposed Models?

9.3.1. Hierarchical Decision-Making Model

The HDM model applied in Chapter 3 and Chapter 4 uses a weighted-averaging operation for the fusion equation. The fusion equation benefits from the performance plots of the previous level's agent(s). If a previous level agent is successful in some decision regions, the rule-base part of the agent to be developed

in the next level has diminished influence in the same decision regions. It is also possible to use a performance plot that makes the decision fusion equation concentrate on a specific decision region. For example, the performance plot can be constructed such that in some decision regions previous level decisions are preserved, and for some specific decision regions the division of labor is separated between the new-rule base to be developed and the previous level decisions. Such a performance plot is show in Fig. 8.1. This performance plot is obtained by changing the performance plot sketched for the sixth level agent in Fig. 3.33 of Chapter 3 (i.e., the success rates in all of the decision regions is assigned as 1 except for a single decision region. The success rate for that decision region is calculated using Equation (3.7)).

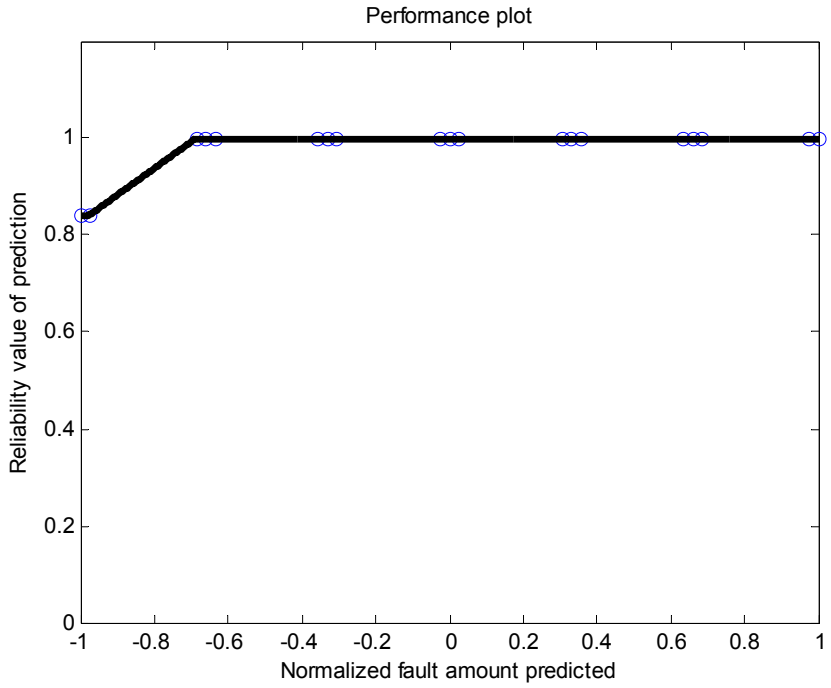

Fig. 9.1: A performance plot.

This performance plot allows an opportunity for the rule-base part to be developed in the agent structure only in a decision region between -1 and -0.66. For the remaining decision region the reliability values of the decisions are equal to 1 (i.e.,

the decisions of previous level agent are assumed to be correct due to the decision fusion equation). This performance plot is a selective one and it tries to correct the decisions only in a specific decision region. This performance plot construction style has two advantageous. Primarily, a selective DM procedure is possible using this performance plot. The second advantage comes from a more interesting fact. Using this performance plot diminishes the computational load, but the advantage depends on the style of selected the fusion equation and the choice of the cost function in the optimization. Let's once again examine the fusion equation:

$$slp = (flp \times rel\_flp) + (nrbp \times (1 - rel\_flp)), \quad (9.1)$$

where 'slp' is the second level prediction, 'flp' is the first level prediction, 'nrbp' is the prediction of the new rule base to be developed, and 'rel_flp' is the reliability value obtained from the performance plot. Using the performance plot in Fig. 9.1, if a first level prediction for a particular input is between -0.66 and 1, the second level prediction for the same input will have the same value as the first level prediction. Furthermore, the prediction for the same input can not be changed due to the rule-base. This situation gives a chance for us to diminish the amount of data that have to be checked in the cost function. Instead of inserting every second level prediction for every input into the cost function, we only have to consider the input whose first level prediction is between -1 and -0.66. Hence, the development of the rule-base can be realized using only a subset of the total input set. This will decrease the computation time to develop the rule-base since the amount of data need to be checked is decreased.

The idea used to construct the performance plots in different manners gives clues about how a selective DM procedure may be possible. This is one of the procedures that can be applied to obtain selective DM procedures. Noise endurance capacity of the model can also be increased using this idea.

The selectivity concept can also be applied for multi-agent applications. Instead of using variable length sequences in the structure of the rule-base parts of the agents,

why do we not choose performance plots having different selectivity characteristics and combine them using a cost function benefiting from these performance plots? Such a structure will increase also the interaction between the agents in consecutive levels. This structure can also be used for multi-criteria DM applications.

9.3.2. Local Decision-Making in Multiple Levels Model

New decentralized methods and clustering techniques can be applied in order to make the model work more successfully. LDM-ML has a flexible structure such that different data evaluation techniques can be applied in the model. Support vector machines or rule-bases with different data evaluation techniques can be used to create local decision makers. Hybrid structures can also be used to represent local decision makers. The hybrid structures can be selected such that the performance of the proposed model can be augmented according to different criteria (i.e., noise endurance, training and validation performance).

9.3.3. Time-Series Data Mining

In general time series data mining methods are used in order to discover or classify patterns that have some special and valuable information in a time series data [53-54]. Several different concepts like time series analysis, genetic algorithms, chaos and nonlinear dynamics and data mining help construction of time series data mining methods [53]. Time series analysis helps understanding of linear time series data. Theoretical background of time series data mining are borrowed from Takens' theorem [55] and Sauer's extension [56] for dynamical systems. If necessary, whenever an optimization process is required, a suitable genetic algorithm or meta-heuristic or probabilistic search method or NN is mostly applied. Two most important concepts in time series data mining are the representation style for the time series data with some pre-processing (normalization, sampling, clustering, distance measure techniques) and obtaining

some temporal patterns from these time series data representations, and extracting some information about the occurrence of events (similarity measure between occurrences of events, creation of some common characters before the event has just happened) depending on some event characterization function based on the satisfaction of an objective function due to clustering or optimization techniques over the temporal patterns [53].

There are many similarities of the time series data mining methods with the facilities used in our DM models. In Chapters 4 and 8, for solving the corresponding case study problems (CS1 and CS2) parts of the data is used in order to represent temporal patterns. Some pre-processing over the selected data is performed (using a distance measure, normalization, clustering) to extract the information about the event occurrence using an optimization process over the structure based on a cost function similar to satisfaction of the objective function. These are the similarities between the methods employed in this thesis and the time series data mining techniques.

Time series data mining techniques can be used as a component of our models (in a level time series data mining techniques can be used to extract the necessary information). However, in circumstances where so many temporal patterns about the time series data is available time series data mining techniques can not be so successful. Besides, what is provided in this thesis is not a family of techniques to be used in DM. Instead, two multi-level DM models which enable using any technique inside these models are introduced.

9.4. Future Studies

It is planned to study on the flexibility aspect of the proposed DM models. Different rule-base structures and hybrid structures will be used and their performance will be monitored in the forthcoming studies. Another study to be examined is the selectivity concept. The ideas explained in Section 9.3.1 will be implemented and their efficiency will be investigated. It is also desired to apply

the proposed models for other kinds of problems associated with different applications and investigate their performances in these new applications.

9.5. Final Remarks

In this dissertation, two different multi-level DM models are proposed. The models possess a framework that can be augmented and improved further. The flexibility of the proposed models is their most advantageous aspects.

REFERENCES

[1] Simon H.A., Dantzig G.B., Hogarth R., Piott C.R., Raiffa H., Schelling T.C., Shepsle K. A., Thaier R., Tversky A., and Winter S., Decision Making and Problem Solving, National Academy Press, Washington DC, 1986.

[2] Turban E., Aronson J. E., and Liang T-P., Decision Support Systems and Intelligent Systems, Prentice Hall, 2005.

[3] Marsh E.R., Slocum A.H., and Otto K.N., Hierarchical Decision Making in Machine Design, Technical report, MIT Precision Engineering Research Center, 1993.

[4] Pasi G., and Yager R.R., Modelling the Concept of Majority Opinion in Group Decision Making, Information Sciences, 176, 390-414, 2006.

[5] Ben-Arieh D., and Easton T., Multi-Criteria Group Consensus under Linear Cost Opinion Elasticity, Decision Support Systems, 43, 713-721, 2007.

[6] Pasi G., and Yager R.R., Modeling the Concept of Fuzzy Majority Opinion, Fuzzy Sets and Systems IFSA 2003 (10^{th} International Fuzzy Systems Association World Congress) Istanbul, Turkey, June 30 – July 2), Springer, Berlin Heidelberg, 143-150, 2003.

[7] Wang Y-M., and Parkan C., Two New Approaches for Assessing the Weights of Fuzzy Opinions in Group Decision Analysis, Information Sciences, 176, 3538-3555, 2006.

[8] Tsiporkova E., and Boeva V., Multi-Step Ranking of Alternatives in a Multi-Criteria and Multi-Expert Decision Making Environment, Information Sciences, 176, 2673-2697, 2006.

[9] Ralescu A.L., Ralescu D.A., and Yamakata Y., Inference by Aggregation of Evidence with Applications to Fuzzy Probabilities, Information Sciences, 177, 378-387, 2007.

[10] Xu Z., Intuitionistic Preference Relations and Their Application in Group Decision Making, Information Sciences, 177, 2363–2379, 2007.

[11] Xu Z-S., and Chen J., An Interactive Method for Multiple Attribute Group Decision Making, Information Sciences, 177, 248-263. 2007.

[12] Martinez L., Liu J., Ruan D., and Yang J-B, Dealing with Heterogeneous Information in Engineering Evaluation Processes, Information Sciences, 177, 1533-1542, 2007.

[13] Li D-F., An Approach to Fuzzy Multi-Attribute Decision Making under Uncertainty, Information Sciences, 169, 97-112, 2005.

[14] Lu J., Shi C., and Zhang G., On Bilevel Multi-Follower Decision Making: General Framework and Solutions, Information Sciences, 176, 1607-1627, 2006.

[15] Pal B.B., and Biswas A., A Fuzzy Multilevel Programming Method for Hierarchical Decision Making, Neural Information Processing: 11th International Conference, ICONIP 2004, Calcutta, India, 904-911, 2004.

[16] Rahman A.F.R., and Fairhurst M.C., A Novel Confidence-Based Framework for Multiple Expert Decision Fusion, Proceedings of the British Machine Vision Conference 1998 (BMVC 1998), Southampton, UK, 1998.

[17] Schuldt A., and Werner S., A Clustering Protocol for Team Formation Based on Concept Location and Time, 5th European Workshop on Multi-Agent Systems (EUMAS 2007), Hammamet, Tunisia, 2007.

[18] Subbu R., Sanderson A.C., Network-Based Distributed Planning Using Coevolutionary Agents: Architecture and Evaluation, IEEE Transactions on Systems, Man, and Cybernetics, Part A, 34(2), 257-269, 2004.

[19] Kahraman C., Büyüközkan G., and Ateş N.Y., A Two Phase Multi-Attribute Decision-Making Approach for New Product Introduction, Information Sciences, 177, 1567-1582, 2007.

[20] Jin B., Tang Y.C., and Zhang Y-Q., Support Vector Machines with Genetic Fuzzy Feature Transformation for Biomedical Data Classification, Information Sciences, 177, 476-489, 2007.

[21] O J., Lee J., Lee J.W., and Zhang B-T., Adaptive Stock Trading with Dynamic Asset Allocation Using Reinforcement Learning, Information Sciences, 176, 2121-2147, 2006.

[22] Pei Z., Resconi G., Wal A.J.V.D., Qin K., and Xu Y., Interpreting and Extracting Fuzzy Decision Rules from Fuzzy Information Systems and Their Inference, Information Sciences, 176, 1869-1897, 2006.

[23] Lee C-S., Jiang C-C., and Hsieh T-C., A Genetic Fuzzy Agent Using Ontology Model for Meeting Scheduling System, Information Sciences, 176, 1131-1155, 2006.

[24] Veeravalli V., Topics in Decentralized Detection, Doctor of Philosophy Thesis in Electrical Engineering, Graduate College of the University of Illinois at Urbana-Champaign, Urbana, Illinois, 1992.

[25] Kılıç E., Fault Detection and Diagnosis in Nonlinear Dynamical Systems- Ph. D Thesis, Middle East Technical University, Ankara, Turkey, 2005.

[26] Pawlak Z., Rough Sets: Theoretical Aspects of Reasoning About Data, Kluwer Academic, Netherlands, Ddordrecht, 1991.

[27] Freund Y., and Schapire R.E., Experiments with a New Boosting Algorithm, Proc. ICML-96, Bari, Italy, 148-156, 1996.

[28] Freund Y., and Schapire R., A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, Journal of Computer System Sciences, 55, 119-139, 1997.

[29] Chen H-Y., Huang C-L., and Fu C-M., Hybrid-Boost Learning for Multi-Pose Face Detection and Facial Expression Recognition, Pattern Recognition, 41, 1173-1185, 2008.

[30] Meynet J., Popovici V., and Thiran J-P., Face Detection with Boosted Gaussian Features, Pattern Recognition, 40, 2283-2291, 2007.

[31] M-Salinas R., G-Silvente M., and Carnicer R.M., Adaptive Multi-Modal Stereo People Tracking without Background Modelling, Journal of Visual Communication and Image Representation, 19, 75–91, 2008.

[32] Grabner H., Nguyen T.T., Gruber B., and Bischof H., On-Line Boosting-Based Car Detection from Aerial Images, ISPRS Journal of Photogrammetry & Remote Sensing, 63, 382-396, 2008.

[33] Zhang C-X., Zhang J-S., and Zhang G-Y, An Efficient Modified Boosting Method for Solving Classification Problems, Journal of Computational and Applied Mathematics, 214, 381–392, 2008.

[34] Zhang C-X., and Zhang J-S., A Local Boosting Algorithm for Solving Classification Problems, Computational Statistics and Data Analysis, 52, 1928–1941, 2008.

[35] Merler S., Caprile B., and Furlanello C., Parallelizing AdaBoost by Weights Dynamics, Computational Statistics and Data Analysis, 51, 2487–2498, 2007.

[36] Li X., Wang L., and Sung E., AdaBoost with SVM-Based Component Classifiers, Engineering Applications of Artificial Intelligence, 21, 785-795, 2008.

[37] Iqbal R.T., Barbu C., and Petry F., Fuzzy Component Based Object Detection, International Journal of Approximate Reasoning, 45, 546–563, 2007.

[38] Viaene S., and G. Dedene, Cost-Sensitive Learning and Decision Making Revisited, European Journal of Operational Research, 166, 212–220, 2005.

[39] Li Y., Yin R-P., Cai Y-Z., and Xu X-M., A New Decision Fusion Method in Support Vector Machine Ensemble, Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 2005.

[40] Basir O., and Yuan X., Engine Fault Diagnosis Based on Multi-Sensor Information Fusion Using Dempster–Shafer Evidence Theory, Information Fusion, 8, 379–386, 2007.

[41] Chiang L.H., Russell E.L., and Braatz R.D., Fault Detection and Diagnosis in Industrial Systems, Springer, London, 2001.

[42] Venkatasubramanian V., Rengaswamy R., Kavuri S.N., and Yin K., A Review of Process Fault Detection and Diagnosis, Part III: Process History Based Methods, Computers and Chemical Engineering, 27, 327–346, 2003.

[43] Mendonça L.F., Sousa J.M.C., and Costa J.M.G.S.D., An Architecture for Fault Detection and Isolation Based on Fuzzy Methods, Expert Systems with Applications, 36, 1092-1104, 2009.

[44] Miguela L.J.D., and Blazquez L.F., Fuzzy Logic-Based Decision-Making for fault Diagnosis in a DC Motor, Engineering Applications of Artificial Intelligence, 18, 423–450, 2005.

[45] Patan K., and Korbicz J., Fault Detection in Catalytic Cracking Converter by Means of Probability Density Approximation, Engineering Applications of Artificial Intelligence, 20 912–923, 2007.

[46] Pisu P., Soliman A. and Rizzoni G., Vehicle Chassis Monitoring System, Control Engineering Practice, 11, 345–354, 2003.

[47] Ting Y., Lu W.B., Chen C.H., and Wang G.K., A Fuzzy Reasoning Design for Fault Detection and Diagnosis of a Computer-Controlled System, Engineering Applications of Artificial Intelligence, 21, 157–170, 2008.

[48] Mitchell T.M., Machine Learning, McGraw-Hill, Singapore, 1997.

[49] Zhu J., Rosset S., Zou H., and Hastie T., Multi-Class Adaboost, Technical Report, 2005.

[50] Hornik K., Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4, 1991, 251-257.

[51] Csaji B.C, Approximation with Artificial Neural Networks, M.Sc. Thesis, Faculty of Science, Eötvös Lorand University, Hungary, 2001.

[52] Beensock M. and Szpire S., Specification Search in Nonlinear Time-Series Models Using the Genetic Algorithm, Journal of Economics and Dynamics Control, 26, 811-835, 2002.

[53] Povinelli R.C., Time Series Data Mining: Identifying Temporal Patterns for Characterization and Prediction of Time series Events, PHD Thesis, Marquette University, Milwaukee, Wisconsin, 1999.

[54] Last M., Kendel A. and Bunke H., Data Mining in Time Series Databases, World Scientific, Series in Machine Perception and Artificial Intelligence, Vol. 57, June 2004.

[55] Takens F., Detecting Strange Attractors in Turbulance, Proceedings of Dynamical Systems and Turbulance, Warwick, 366-381, 1980.

[56] Sauer T., Yorke J.A. and Casdagli M.A., Embedology, Journal of Statistical Physics, 65, 579-616, 1991.

APPENDIX A

THE TRAINING AND THE VALIDATION SCENARIOS

Table A.1: The fault characters in the training scenarios: 't_1', 't_2', 't_3' and 't_4' correspond to the seconds the faults are first initiated. 'Fault_T_1', 'Fault_T_2', 'Fault_T_3', and 'Fault_T_4' represent the normalized fault amounts created at respective seconds which are unchanged after they are generated.

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
1	0.33	3	-	-	-	-	-	-
2	-	-	0.33	2	-	-	-	-
3	-	-	-	-	0.33	4	-	-
4	-	-	-	-	-	-	0.33	6
5	0.66	2	-	-	-	-	-	-
6	-	-	0.66	3	-	-	-	-
7	-	-	-	-	0.66	1	-	-
8	-	-	-	-	-	-	0.66	4
9	1	7	-	-	-	-	-	-
10	-	-	1	2	-	-	-	-
11	-	-	-	-	1	3	-	-
12	-	-	-	-	-	-	1	1
13	0.33	5	-	-	-	-	-	-
14	-	-	0.33	4	-	-	-	-
15	-	-	-	-	0.33	3	-	-
16	-	-	-	-	-	-	0.33	1
17	0.66	7	-	-	-	-	-	-
18	-	-	0.66	4	-	-	-	-
19	-	-	-	-	0.66	6	-	-
20	-	-	-	-	-	-	0.66	8
21	1	4	-	-	-	-	-	-
22	-	-	1	1	-	-	-	-

Table A.1 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
23	-	-	-	-	1	6	-	-
24	-	-	-	-	-	-	1	5
25	1	7	0.66	3	-	-	-	-
26	0.33	3	-	-	0.66	5	-	-
27	0.66	2	-	-	-	-	0.33	1
28	-	-	1	4	1	3	-	-
29	-	-	0.66	5	-	-	0.33	8
30	-	-	-	-	0.66	2	0.33	2
31	0.66	6	0.33	7	-	-	-	-
32	0.33	9	-	-	0.66	2	-	-
33	1	7	-	-	-	-	1	3
34	-	-	0.66	6	0.33	7	-	-
35	-	-	0.33	3	-	-	0.66	4
36	-	-	-	-	1	8	0.33	1
37	0.33	4	0.66	5	0.33	2	-	-
38	1	7	1	4	-	-	0.66	1
39	0.66	2	-	-	0.66	5	0.66	5
40	-	-	0.33	8	0.33	9	0.33	3
41	0.33	3	0.66	7	1	3	-	-
42	1	7	0.33	6	-	-	0.66	7
43	0.33	2	-	-	0.33	1	0.66	1
44	-	-	0.33	4	1	9	1	9
45	0.66	2	0.66	2	0.66	1	-	-
46	0.66	5	0.33	7	-	-	1	6
47	1	1	-	-	0.66	5	0.33	9
48	-	-	1	7	0.66	7	0.33	7
49	1	6	0.66	4	0.66	8	0.33	9
50	0.33	3	0.33	7	0.66	8	0.66	9
51	0.66	1	0.66	1	1	1	1	1
52	0.33	9	0.33	8	0.33	9	0.33	3
53	0.33	7	1	6	0.66	8	0.66	4
54	0.33	6	1	1	1	9	1	5
55	0.66	3	0.33	6	0.33	9	0.33	6

Table A.1 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
56	1	1	1	1	1	2	0.33	1
57	1	6	0.33	3	0.66	3	1	3
58	0.33	7	0.66	8	0.66	4	0.66	4
59	1	8	1	7	0.66	5	1	8
60	1	5	0.33	2	0.66	7	0.66	2
61	-1	3	-	-	-	-	-	-
62	-	-	-1	2	-	-	-	-
63	-	-	-	-	-1	4	-	-
64	-	-	-	-	-	-	-1	6
65	-0.66	2	-	-	-	-	-	-
66	-	-	-0.66	3	-	-	-	-
67	-	-	-	-	-0.66	1	-	-
68	-	-	-	-	-	-	-0.66	4
69	-0.33	7	-	-	-	-	-	-
70	-	-	-0.33	2	-	-	-	-
71	-	-	-	-	-0.33	3	-	-
72	-	-	-	-	-	-	-0.33	1
73	-1	5	-	-	-	-	-	-
74	-	-	-1	4	-	-	-	-
75	-	-	-	-	-1	3	-	-
76	-	-	-	-	-	-	-1	1
77	-0.66	7	-	-	-	-	-	-
78	-	-	-0.66	4	-	-	-	-
79	-	-	-	-	-0.66	6	-	-
80	-	-	-	-	-	-	-0.66	8
81	-0.33	4	-	-	-	-	-	-
82	-	-	-0.33	1	-	-	-	-
83	-	-	-	-	-0.33	6	-	-
84	-	-	-	-	-	-	-0.33	5
85	-0.33	7	-0.66	3	-	-	-	-
86	-1	3	-	-	-0.66	5	-	-
87	-0.66	2	-	-	-	-	-1	1
88	-	-	-0.33	4	-0.33	3	-	-

Table A.1 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
89	-	-	-0.66	5	-	-	-1	8
90	-	-	-	-	-0.66	2	-1	2
91	-0.66	6	-1	7	-	-	-	-
92	-1	9	-	-	-0.66	2	-	-
93	-0.33	7	-	-	-	-	-0.33	3
94	-	-	-0.66	6	-1	7	-	-
95	-	-	-1	3	-	-	-0.66	4
96	-	-	-	-	-0.33	8	-1	1
97	-1	4	-0.66	5	-1	2	-	-
98	-0.33	7	-0.33	4	-	-	-0.66	1
99	-0.66	2	-	-	-0.66	5	-0.66	5
100	-	-	-1	8	-1	9	-1	3
101	-1	3	-0.66	7	-0.33	3	-	-
102	-0.33	7	-1	6	-	-	-0.66	7
103	-1	2	-	-	-1	1	-0.66	1
104	-	-	-1	4	-0.33	9	-0.33	9
105	-0.66	2	-0.66	2	-0.66	1	-	-
106	-0.66	5	-1	7	-	-	-0.33	6
107	-0.33	1	-	-	-0.66	5	-1	9
108	-	-	-0.33	7	-0.66	7	-1	7
109	-0.33	6	-0.66	4	-0.66	8	-1	9
110	-1	3	-1	7	-0.66	8	-0.66	9
111	-0.66	1	-0.66	1	-0.33	1	-0.33	1
112	-1	9	-1	8	-1	9	-1	3
113	-1	7	-0.33	6	-0.66	8	-0.66	4
114	-1	6	-0.33	1	-0.33	9	-0.33	5
115	-0.66	3	-1	6	-1	9	-1	6
116	-0.33	1	-0.33	1	-0.33	2	-1	1
117	-0.33	6	-1	3	-0.66	3	-0.33	3
118	-1	7	-0.66	8	-0.66	4	-0.66	4
119	-0.33	8	1	7	-0.66	5	-0.33	8
120	-0.33	5	-1	2	-0.66	7	-0.66	2
121	-0.33	5	1	2	-	-	-	-
122	-1	6	-	-	0.33	1	-	-

Table A.1 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
123	-0.66	9	-	-	-	-	0.66	1
124	-	-	-1	5	0.66	2	-	-
125	-	-	-0.66	6	-	-	1	7
126	-	-	-	-	-0.33	9	1	5
127	0.66	4	-1	7	-	-	-	-
128	1	8	-	-	-0.66	5	-	-
129	0.33	7	-	-	-	-	-0.33	9
130	-	-	0.66	5	-0.33	3	-	-
131	-	-	0.33	9	-	-	-0.66	4
132	-	-	-	-	1	2	-1	2
133	-0.66	7	-1	5	1	5	-	-
134	-1	8	0.33	9	-1	5	-	-
135	-0.33	7	-1	2	-	-	1	4
136	-0.66	4	1	2	-	-	0.66	5
137	-1	2	-	-	-0.66	9	1	4
138	-0.33	3	-	-	1	8	-1	1
139	-0.66	5	1	7	1	2	-	-
140	-0.33	2	0.33	1	-	-	0.66	9
141	-1	9	-	-	0.33	9	0.33	9
142	-	-	-0.33	1	-0.33	1	1	1
143	-	-	-1	5	0.33	3	-1	2
144	-	-	-0.66	6	0.33	2	0.66	8
145	0.33	4	1	9	-1	8	-	-
146	0.66	3	-0.66	8	0.66	6	-	-
147	1	3	0.66	5	-	-	-1	9
148	0.33	3	-0.33	7	-	-	0.66	5
149	0.66	1	-	-	0.33	1	-1	1
150	1	9	-	-	-1	8	0.66	1
151	1	5	-1	5	-1	5	-	-
152	0.33	7	-0.66	8	-	-	-0.33	9
153	0.66	1	-	-	-0.33	2	-0.66	3
154	-	-	0.66	4	1	4	-0.33	8
155	-	-	0.66	3	-0.66	3	1	9
156	-	-	1	3	-0.66	5	-0.33	7

Table A.1 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
157	-1	3	-1	5	-0.33	7	0.66	7
158	-0.66	4	-0.33	2	0.66	7	-0.33	5
159	-0.33	2	-1	8	1	4	1	9
160	-0.33	8	1	8	-1	9	-0.66	9
161	-0.33	1	0.66	2	-0.33	3	1	4
162	-1	7	1	6	0.33	5	-1	6
163	-0.66	9	0.33	1	0.66	8	1	2
164	1	2	-0.66	8	-0.33	9	0.33	1
165	0.33	8	-1	1	1	3	-0.66	8
166	0.66	2	-0.33	2	0.33	2	0.33	2
167	1	9	1	4	-1	2	-1	6
168	0.66	4	0.33	7	-0.66	6	0.66	5
169	1	9	1	6	1	3	-1	1
170	0.33	1	-1	1	-1	1	-1	1

Table A.2: The created fault characters in the first set of validation scenarios: 't_1', 't_2', 't_3' and 't_4' correspond to the seconds the faults are first initiated. '$Fault\_T_1$', '$Fault\_T_2$', '$Fault\_T_3$', and '$Fault\_T_4$' represent the normalized fault amounts created at respective seconds which are unchanged after they are generated.

Scenario	$Fault\_T_1$	t_1	$Fault\_T_2$	t_2	$Fault\_T_3$	t_3	$Fault\_T_4$	t_4
1	0.33	5	-	-	-	-	-	-
2	-	-	0.33	4	-	-	-	-
3	-	-	-	-	0.33	9	-	-
4	-	-	-	-	-	-	0.33	1
5	0.66	7	-	-	-	-	-	-
6	-	-	0.66	1	-	-	-	-
7	-	-	-	-	0.66	8	-	-
8	-	-	-	-	-	-	0.66	9
9	1	6	-	-	-	-	-	-
10	-	-	1	7	-	-	-	-
11	-	-	-	-	1	9	-	-
12	-	-	-	-	-	-	1	1
13	0.33	1	-	-	-	-	-	-
14	-	-	0.33	3	-	-	-	-
15	-	-	-	-	0.33	7	-	-
16	-	-	-	-	-	-	0.33	8
17	0.66	9	-	-	-	-	-	-
18	-	-	0.66	3	-	-	-	-
19	-	-	-	-	0.66	1	-	-
20	-	-	-	-	-	-	0.66	7
21	1	7	-	-	-	-	-	-
22	-	-	1	6	-	-	-	-
23	-	-	-	-	1	2	-	-
24	-	-	-	-	-	-	1	9
25	1	5	0.66	8	-	-	-	-
26	0.33	4	-	-	0.66	3	-	-
27	0.66	7	-	-	-	-	0.33	2
28	-	-	1	5	1	7	-	-
29	-	-	0.66	1	-	-	0.33	1
30	-	-	-	-	0.66	8	0.33	6

Table A.2 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
31	0.66	3	0.33	3	-	-	-	-
32	0.33	1	-	-	0.66	9	-	-
33	1	6	-	-	-	-	1	5
34	-	-	0.66	6	0.33	5	-	-
35	-	-	0.33	9	-	-	0.66	4
36	-	-	-	-	1	6	0.33	7
37	0.33	6	0.66	6	0.33	3	-	-
38	1	4	1	5	-	-	0.66	3
39	0.66	4	-	-	0.66	6	0.66	8
40	-	-	0.33	7	0.33	3	0.33	1
41	0.33	1	0.66	2	1	4	-	-
42	1	8	0.33	9	-	-	0.66	1
43	0.33	2	-	-	0.33	7	0.66	6
44	-	-	0.33	9	1	8	1	7
45	0.66	5	0.66	1	0.66	7	-	-
46	0.66	6	0.33	3	-	-	1	3
47	1	4	-	-	0.66	7	0.33	3
48	-	-	1	6	0.66	8	0.33	2
49	1	8	0.66	7	0.66	6	0.33	5
50	0.33	2	0.33	1	0.66	4	0.66	8
51	0.66	1	0.66	1	1	1	1	1
52	0.33	3	0.33	2	0.33	1	0.33	7
53	0.33	5	1	9	0.66	2	0.66	6
54	0.33	7	1	3	1	8	1	2
55	0.66	4	0.33	6	0.33	7	0.33	1
56	1	3	1	7	1	8	0.33	7
57	1	3	0.33	9	0.66	6	1	7
58	0.33	6	0.66	4	0.66	2	0.66	7
59	1	9	1	2	0.66	1	1	5
60	1	3	0.33	3	0.66	6	0.66	3
61	-1	3	-	-	-	-	-	-
62	-	-	-1	2	-	-	-	-
63	-	-	-	-	-1	8	-	-
64	-	-	-	-	-	-	-1	3

Table A.2 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
65	-0.66	8	-	-	-	-	-	-
66	-	-	-0.66	6	-	-	-	-
67	-	-	-	-	-0.66	4	-	-
68	-	-	-	-	-	-	-0.66	7
69	-0.33	1	-	-	-	-	-	-
70	-	-	-0.33	9	-	-	-	-
71	-	-	-	-	-0.33	2	-	-
72	-	-	-	-	-	-	-0.33	5
73	-1	7	-	-	-	-	-	-
74	-	-	-1	9	-	-	-	-
75	-	-	-	-	-1	4	-	-
76	-	-	-	-	-	-	-1	7
77	-0.66	2	-	-	-	-	-	-
78	-	-	-0.66	7	-	-	-	-
79	-	-	-	-	-0.66	3	-	-
80	-	-	-	-	-	-	-0.66	2
81	-0.33	6	-	-	-	-	-	-
82	-	-	-0.33	8	-	-	-	-
83	-	-	-	-	-0.33	3	-	-
84	-	-	-	-	-	-	-0.33	7
85	-0.33	6	-0.66	7	-	-	-	-
86	-1	4	-	-	-0.66	4	-	-
87	-0.66	5	-	-	-	-	-1	7
88	-	-	-0.33	8	-0.33	2	-	-
89	-	-	-0.66	8	-	-	-1	3
90	-	-	-	-	-0.66	4	-1	7
91	-0.66	8	-1	2	-	-	-	-
92	-1	6	-	-	-0.66	7	-	-
93	-0.33	8	-	-	-	-	-0.33	4
94	-	-	-0.66	3	-1	5	-	-
95	-	-	-1	7	-	-	-0.66	3
96	-	-	-	-	-0.33	2	-1	4
97	-1	3	-0.66	6	-1	7	-	-
98	-0.33	1	-0.33	2	-	-	-0.66	4

Table A.2 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
99	-0.66	9	-	-	-0.66	4	-0.66	9
100	-	-	-1	5	-1	8	-1	6
101	-1	2	-0.66	8	-0.33	8	-	-
102	-0.33	3	-1	3	-	-	-0.66	3
103	-1	2	-	-	-1	1	-0.66	1
104	-	-	-1	5	-0.33	7	-0.33	1
105	-0.66	5	-0.66	5	-0.66	8	-	-
106	-0.66	8	-1	3	-	-	-0.33	1
107	-0.33	5	-	-	-0.66	6	-1	1
108	-	-	-0.33	3	-0.66	3	-1	3
109	-0.33	9	-0.66	2	-0.66	1	-1	6
110	-1	4	-1	3	-0.66	6	-0.66	2
111	-0.66	8	-0.66	4	-0.33	5	-0.33	1
112	-1	3	-1	3	-1	2	-1	3
113	-1	4	-0.33	9	-0.66	9	-0.66	1
114	-1	6	-0.33	6	-0.33	8	-0.33	3
115	-0.66	8	-1	9	-1	1	-1	1
116	-0.33	1	-0.33	1	-0.33	2	-1	1
117	-0.33	8	-1	2	-0.66	8	-0.33	4
118	-1	6	-0.66	3	-0.66	9	-0.66	1
119	-0.33	2	1	3	-0.66	7	-0.33	5
120	-0.33	6	-1	6	-0.66	2	-0.66	7
121	-0.33	4	1	1	-	-	-	-
122	-1	3	-	-	0.33	2	-	-
123	-0.66	8	-	-	-	-	0.66	6
124	-	-	-1	7	0.66	5	-	-
125	-	-	-0.66	3	-	-	1	3
126	-	-	-	-	-0.33	4	1	8
127	0.66	9	-1	4	-	-	-	-
128	1	7	-	-	-0.66	4	-	-
129	0.33	6	-	-	-	-	-0.33	5
130	-	-	0.66	8	-0.33	1	-	-
131	-	-	0.33	4	-	-	-0.66	9
132	-	-	-	-	1	6	-1	4

Table A.2 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
133	-0.66	6	-1	4	1	3	-	-
134	-1	7	0.33	9	-1	9	-	-
135	-0.33	6	-1	4	-	-	1	6
136	-0.66	3	1	4	-	-	0.66	8
137	-1	4	-	-	-0.66	8	1	4
138	-0.33	3	-	-	1	8	-1	1
139	-0.66	9	1	9	1	9	-	-
140	-0.33	2	0.33	2	-	-	0.66	2
141	-1	5	-	-	0.33	8	0.33	5
142	-	-	-0.33	1	-0.33	1	1	1
143	-	-	-1	4	0.33	5	-1	3
144	-	-	-0.66	5	0.33	2	0.66	5
145	0.33	6	1	4	-1	1	-	-
146	0.66	7	-0.66	3	0.66	2	-	-
147	1	8	0.66	3	-	-	-1	4
148	0.33	9	-0.33	9	-	-	0.66	1
149	0.66	3	-	-	0.33	3	-1	8
150	1	9	-	-	-1	9	0.66	9
151	1	5	-1	5	-1	5	-	-
152	0.33	4	-0.66	3	-	-	-0.33	1
153	0.66	5	-	-	-0.33	9	-0.66	4
154	-	-	0.66	6	1	2	-0.33	2
155	-	-	0.66	8	-0.66	5	1	3
156	-	-	1	9	-0.66	6	-0.33	2
157	-1	2	-1	6	-0.33	5	0.66	9
158	-0.66	3	-0.33	7	0.66	7	-0.33	1
159	-0.33	9	-1	2	1	4	1	8
160	-0.33	3	1	6	-1	9	-0.66	7
161	-0.33	4	0.66	4	-0.33	4	1	1
162	-1	8	1	7	0.33	6	-1	5
163	-0.66	3	0.33	1	0.66	2	1	9
164	1	3	-0.66	4	-0.33	3	0.33	5
165	0.33	4	-1	3	1	3	-0.66	6
166	0.66	9	-0.33	9	0.33	4	0.33	4

Table A.2 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
167	1	9	1	4	-1	2	-1	6
168	0.66	6	0.33	5	-0.66	7	0.66	2
169	1	3	1	6	1	3	-1	2
170	0.33	1	-1	2	-1	1	-1	2

Table A.3: The created fault characters in the second set of validation scenarios: 't_1', 't_2', 't_3' and 't_4' correspond to the seconds the faults are first initiated. '$Fault\_T_1$', '$Fault\_T_2$', '$Fault\_T_3$', and '$Fault\_T_4$' represent the normalized fault amounts created at respective seconds which are unchanged after they are generated.

Scenario	$Fault\_T_1$	t_1	$Fault\_T_2$	t_2	$Fault\_T_3$	t_3	$Fault\_T_4$	t_4
1	0.30	3	-	-	-	-	-	-
2	-	-	0.30	2	-	-	-	-
3	-	-	-	-	0.30	4	-	-
4	-	-	-	-	-	-	0.30	6
5	0.63	2	-	-	-	-	-	-
6	-	-	0.63	3	-	-	-	-
7	-	-	-	-	0.63	1	-	-
8	-	-	-	-	-	-	0.63	4
9	0.97	7	-	-	-	-	-	-
10	-	-	0.97	2	-	-	-	-
11	-	-	-	-	0.97	3	-	-
12	-	-	-	-	-	-	1	1
13	0.36	5	-	-	-	-	-	-
14	-	-	0.36	4	-	-	-	-
15	-	-	-	-	0.36	3	-	-
16	-	-	-	-	-	-	0.36	1
17	0.69	7	-	-	-	-	-	-
18	-	-	0.69	4	-	-	-	-
19	-	-	-	-	0.69	6	-	-
20	-	-	-	-	-	-	0.69	8
21	0.94	4	-	-	-	-	-	-
22	-	-	0.94	1	-	-	-	-
23	-	-	-	-	0.94	6	-	-
24	-	-	-	-	-	-	0.94	5
25	0.97	7	0.63	3	-	-	-	-
26	0.3	3	-	-	0.63)	5	-	-
27	0.63	2	-	-	-	-	0.3	1
28	-	-	0.97	4	0.97	3	-	-
29	-	-	0.63	5	-	-	0.3	8
30	-	-	-	-	0.63	2	0.3	2

Table A.3 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
31	0.69	6	0.36	7	-	-	-	-
32	0.36	9	-	-	0.69	2	-	-
33	0.94	7	-	-	-	-	0.94	3
34	-	-	0.69	6	0.36	7	-	-
35	-	-	0.36	3	-	-	0.69	4
36	-	-	-	-	0.94	8	0.36	1
37	0.3	4	0.63	5	0.3	2	-	-
38	0.97	7	0.97	4	-	-	0.63	1
39	0.63	2	-	-	0.63	5	0.63	5
40	-	-	0.3	8	0.3	9	0.3	3
41	0.36	3	0.69	7	0.94	3	-	-
42	0.94	7	0.36	6	-	-	0.69	7
43	0.36	2	-	-	0.36	1	0.69	1
44	-	-	0.36	4	0.94	9	0.94	9
45	0.69	2	0.63	2	0.63	1	-	-
46	0.63	5	0.36	7	-	-	0.94	6
47	0.94	1	-	-	0.63	5	0.30	9
49	0.97	6	0.69	4	0.69	8	0.36	9
50	0.36	3	0.36	7	0.69	8	0.69	9
51	0.66	1	0.66	1	1	1	1	1
52	0.36	9	0.36	8	0.36	9	0.36	3
53	0.30	7	0.94	6	0.63	8	0.63	4
54	0.30	6	0.94	1	0.94	9	0.94	5
55	0.63	3	0.30	6	0.30	9	0.30	6
56	0.94	1	0.94	1	0.94	2	0.30	1
57	0.94	6	0.36	3	0.69	3	0.97	3
58	0.36	7	0.63	8	0.63	4	0.69	4
59	0.94	8	0.36	7	0.63	5	0.97	8
60	0.97	5	0.3	2	0.63	7	0.69	2
61	-1	3	-	-	-	-	-	-
62	-	-	-1	2	-	-	-	-
63	-	-	-	-	-0.97	4	-	-
64	-	-	-	-	-	-	-0.97	6

Table A.3 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
65	-0.63	2	-	-	-	-	-	-
66	-	-	-0.63	3	-	-	-	-
67	-	-	-	-	-0.63	1	-	-
68	-	-	-	-	-	-	-0.63	4
69	-0.3	7	-	-	-	-	-	-
70	-	-	-0.3	2	-	-	-	-
71	-	-	-	-	-0.3	3	-	-
72	-	-	-	-	-	-	-0.3	1
73	-0.94	5	-	-	-	-	-	-
74	-	-	-0.94	4	-	-	-	-
75	-	-	-	-	-0.94	3	-	-
76	-	-	-	-	-	-	-0.94	1
77	-0.69	7	-	-	-	-	-	-
78	-	-	-0.69	4	-	-	-	-
79	-	-	-	-	-0.69	6	-	-
80	-	-	-	-	-	-	-0.69	8
81	-0.36	4	-	-	-	-	-	-
82	-	-	-0.36	1	-	-	-	-
83	-	-	-	-	-0.36	6	-	-
84	-	-	-	-	-	-	-0.36	5
85	-0.30	7	-0.63	3	-	-	-	-
86	-0.97	3	-	-	-0.63	5	-	-
87	-0.63	2	-	-	-	-	-0.97	1
88	-	-	-0.3	4	-0.3	3	-	-
89	-	-	-0.63	5	-	-	-0.97	8
90	-	-	-	-	-0.63	2	-0.97	2
91	-0.69	6	-0.94	7	-	-	-	-
92	-0.94	9	-	-	-0.69	2	-	-
93	-0.36	7	-	-	-	-	-0.36	3
94	-	-	-0.69	6	-0.94	7	-	-
95	-	-	-0.94	3	-	-	-0.69	4
96	-	-	-	-	-0.36	8	-0.94	1
97	-0.97	4	-0.63	5	-0.97	2	-	-
98	-0.3	7	-0.3	4	-	-	-0.63	1

Table A.3 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
99	-0.63	2	-	-	-0.63	5	-0.63	5
100	-	-	-0.97	8	-0.97	9	-0.97	3
101	-0.94	3	-0.69	7	-0.36	3	-	-
102	-0.36	7	-0.94	6	-	-	-0.69	7
103	-1	2	-	-	-1	1	-0.66	1
104	-	-	-0.94	4	-0.36	9	-0.36	9
105	-0.69	2	-0.63	2	-0.69	1	-	-
106	-0.63	5	-0.94	7	-	-	-0.36	6
107	-0.3	1	-	-	-0.63	5	-0.97	9
108	-	-	-0.36	7	-0.69	7	-0.97	7
109	-0.36	6	-0.69	4	-0.69	8	-0.94	9
110	-0.97	3	-0.97	7	-0.63	8	-0.63	9
111	-0.63	1	-0.69	1	-0.36	1	-0.36	1
112	-0.94	9	-0.97	8	-0.97	9	-0.97	3
113	-0.97	7	-0.3	6	-0.69	8	-0.69	4
114	-0.94	6	-0.36	1	-0.3	9	-0.3	5
115	-0.63	3	-0.97	6	-0.97	9	-0.94	6
116	-0.33	1	-0.33	1	-0.33	2	-1	1
117	-0.36	6	-0.94	3	-0.69	3	-0.33	3
118	-0.97	7	-0.69	8	-0.63	4	-0.69	4
119	-0.36	8	0.97	7	-0.69	5	-0.3	8
120	-0.3	5	-0.94	2	-0.69	7	-0.63	2
121	-0.3	5	0.97	2	-	-	-	-
122	-0.97	6	-	-	0.36	1	-	-
123	-0.69	9	-	-	-	-	0.69	1
124	-	-	-0.94	5	0.63	2	-	-
125	-	-	-0.63	6	-	-	0.97	7
126	-	-	-	-	-0.3	9	0.94	5
127	0.69	4	-0.94	7	-	-	-	-
128	0.97	8	-	-	-0.69	5	-	-
129	0.3	7	-	-	-	-	-0.3	9
130	-	-	0.69	5	-0.3	3	-	-
131	-	-	0.36	9	-	-	-0.69	4
132	-	-	-	-	0.97	2	-0.94	2

Table A.3 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
133	-0.63	7	-0.97	5	0.94	5	-	-
134	-0.97	8	0.3	9	-0.97	5	-	-
135	-0.3	7	-0.94	2	-	-	0.94	4
136	-0.69	4	0.97	2	-	-	-0.36	5
137	-0.94	2	-	-	-0.69	9	0.94	4
138	-0.33	3	-	-	1	8	-1	1
139	-0.63	5	0.94	7	0.94	2	-	-
140	-0.3	2	0.36	1	-	-	0.63	9
141	-0.94	9	-	-	0.36	9	0.36	9
142	-	-	-0.33	1	-0.33	1	1	1
143	-	-	-0.93	5	0.3	3	-0.97	2
144	-	-	-0.69	6	0.36	2	0.69	8
145	0.36	4	0.94	9	-0.97	8	-	-
146	0.69	3	-0.69	8	0.69	6	-	-
147	0.94	3	0.63	5	-	-	-0.97	9
148	0.3	3	-0.36	7	-	-	0.63	5
149	0.63	1	-	-	0.3	1	-0.97	1
150	0.97	9	-	-	-0.94	8	0.69	1
151	1	5	-1	5	-1	5	-	-
152	0.36	7	-0.63	8	-	-	-0.36	9
153	0.63	1	-	-	-0.3	2	-0.63	3
154	-	-	0.69	4	0.94	4	-0.3	8
155	-	-	0.69	3	-0.69	3	0.94	9
156	-	-	0.94	3	-0.63	5	-0.3	7
157	-0.97	3	-0.97	5	-0.3	7	0.69	7
158	-0.63	4	-0.3	2	0.63	7	-0.3	5
159	-0.3	2	-0.97	8	0.97	4	0.97	9
160	-0.36	8	0.97	8	-0.94	9	-0.69	9
161	-0.36	1	0.69	2	-0.36	3	0.94	4
162	-0.97	7	0.97	6	0.36	5	-0.94	6
163	-0.69	9	0.36	1	0.69	8	0.94	2
164	0.97	2	-0.69	8	-0.3	9	0.36	1
165	0.36	8	-0.97	1	0.97	3	-0.63	8
166	0.63	2	-0.3	2	0.3	2	0.3	2

Table A.3 (continued)

Scenario	Fault_T_1	t_1	Fault_T_2	t_2	Fault_T_3	t_3	Fault_T_4	t_4
167	1	9	1	4	-1	2	-1	6
168	0.63	4	0.3	7	-0.69	6	0.63	5
169	0.94	9	0.94	6	0.94	3	-0.97	1
170	0.36	1	-0.94	1	-0.94	1	-0.94	1

APPENDIX B

THE UNIVERSAL APPROXIMATION THEOREM

Let '$\vartheta(.)$' be an arbitrary activation function. Let $X \subseteq R^m$ and 'X' is compact. The space of continuous functions on 'X' is denoted by '$C(X)$', then $\forall f \in C(X)$, $\forall \varepsilon > 0$: $\exists n \in N$, $a_{ij}, b_i, w_i \in R$, $i \in \{1...n\}$, $j \in \{1...m\}$ which is an artificial neural network structure with one hidden layer whose output is obtained as in Equation (B.1):

$$(A_n f)(x_1,....,x_m) = \sum_{i=1}^{n} w_i \varphi\left(\sum_{j=1}^{m} a_{ij} x_j + b_i\right) \tag{B.1}$$

and as an approximation of the function '$f(.)$' one can write (B.2);

$$\|f - A_n f\| < \varepsilon \tag{B.2}$$

(In the notation '$A_n f$', 'n' shows the number of hidden layer neurons and '$A_n f$' is the output of the neural network).

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name	:	Beldek, Ulaş
Nationality	:	Turkish (TC)
Date and Place of Birth	:	20 April 1977, Mersin
Marital Status	:	Single
Phone	:	+90 312 284 45 00 / 302
Fax	:	+90 312 210 22 91
e-mail	:	u.beldek@cankaya.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU, Elec-Electr. Eng. Dept.	2001
BS	METU, Elec-Electr. Eng. Dept.	1999
High School	İçel Anadolu Lisesi, Mersin	1995

WORK EXPERIENCE

Year	Place	Enrollment
2002- Present	Çankaya University	Lecturer
1999-2002	METU, Elec-Electr. Eng. Dept.	Assistant
1998 July	IZOCAM	Intern Engineering Student
1997 August	ÇİMSA	Intern Engineering Student

FOREIGN LANGUAGES

English (advanced level).

PUBLICATIONS

1- **U. Beldek**, K. Leblebicioğlu, 2007, Strategy Creation, Decomposition and Distribution in Particle Navigation, **Information Sciences**, 177: 3, 755-770.

2**- U. Beldek**, K. Leblebicioğlu, 2009, Local Decision Making and Decision Fusion in Hierarchical Levels, **TOP**: An Official Journal of the Spanish Society of Statistics and Operations Research**,** 17, 44-69.

HOBBIES

Swimming, Reading, Cinema.