University of Alberta

Application of Computational Intelligence Techniques to Stress Prediction on Earthmoving Equipment

by

Samer Sanduga 

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Electrical and Computer Engineering

Edmonton, Alberta
Fall 2008

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-47405-1
Our file Notre référence
ISBN: 978-0-494-47405-1

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

In this research, computational intelligence techniques were used to predict the forces distribution across the wheels of a bulldozer, when levelling overburden on a dump site. This is accomplished by modeling the volume of overburden the bulldozer is pushing and summing the moments around the front and rear wheel force. Determining the forces at the wheels of the bulldozer will allow us to design a system that will reduce equipment downtime due to uneven wearing of the wheels.

Neurofuzzy systems were chosen to model relations between measured quantities and overburden. This type of models ensures easiness of learning, as well as high transparency. Structures of neurofuzzy systems were determined using a genetic programming based system. The experiments were conducted for two data sets collected in winter and early spring. IF-THEN rules were extracted from both models. The rules indicated several fuzzy terms as important, with the most important being the rimpull force.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mining industry relies heavily on its fleets of tracks, shovels, and support equipment to meet its mining operational objectives. Maintenance and downtime reduction of the equipment is especially important for the mining operation. The fact that this equipment is in operation 24/7 means that the scale of these problems, their distinctiveness and impact on everyday business of mines are of paramount importance. Any improvements of effectiveness of operations of the equipment make a notable impact on the mining productivity and the plant-wide profits. For these reasons, there is a genuine and evident need in the surface mining industry to develop a better understanding of different factors impacting work and life span of the earth moving equipment.

1.1 Industry Background

Bulldozers and motor graders are one of the principal machinery used in the surface mining. They perform numerous operations such as pushing, pulling or ripping and maintain of road by motor graders. Bulldozers components are constantly exposed to extensive and long lasting forces that cause their substantial wear. In order to keep bulldozers fully functional, there is a need for a constant monitoring of conditions of the components and their frequent replacement. It is then essential to understand how forces are distributed among different components and how they affect different parts of the components. Such understanding would help identify components with the highest exposure to forces, and help determine the best method to ensure a uniform distribution of forces.

1.2 Motivation

A distribution of mechanical forces is constantly changing during operations of a bulldozer. These changes are associated with continuous changes in bulldozer's load, working conditions, operator habits.
There are many factors influencing this distribution, therefore it seems that the best approach to gain some knowledge about forces is a comprehensive measurement process leading to collection of data representing a number of different parameters associated with operating conditions of bulldozer.

In such circumstances, the first step in an estimation process of force distribution is analysis of the collected data. There are several interesting and practical questions one could pose that really motivate a thorough analysis of the data. For instance, one could be interested in learning about some major relationships and groups of data that could be distinguished in the dataset. What do they represent in terms of the working conditions of the equipment? How are they interrelated? Do they offer any meaningful insight into the underlying phenomenon that generated the data themselves? Do they offer any significant insight into the nature of the processes and eventually help draw some conclusions about the reliability of the equipment and improve their working conditions? All of those are relevant questions for which a comprehensive data analysis could offer some useful answers.

1.3 Problem description

In order to predict the force distribution across the wheels of the bulldozer, we sum the moments of all forces acting on the bulldozer. Once the moments are summed up, we can calculate the forces at locations F_f and F_r (front force, and rear force), Figure 1.

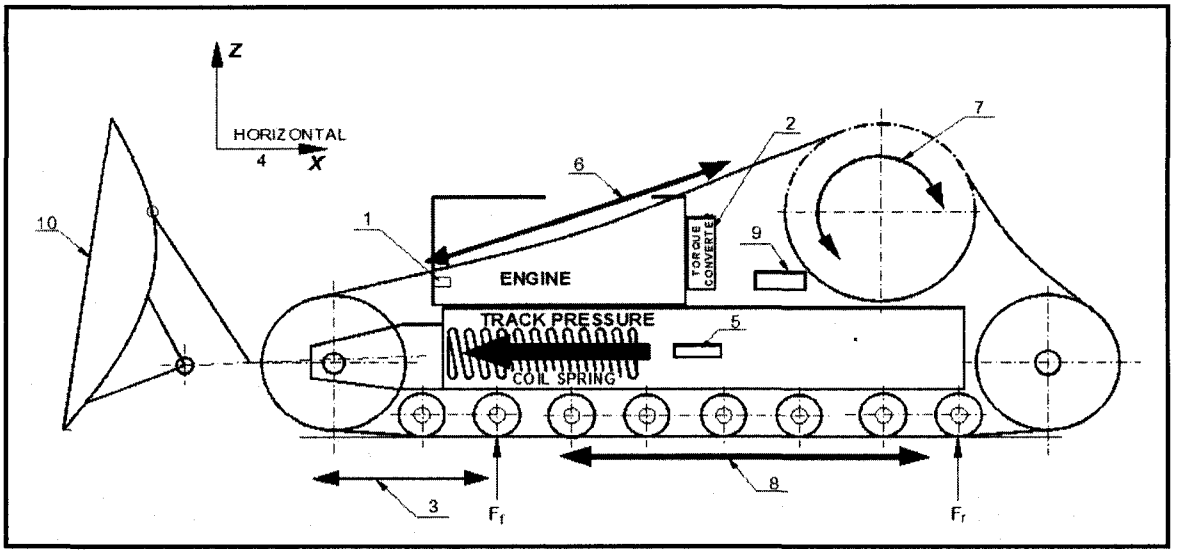


Figure 1: Forces acting on the bulldozer wheels

Because the observed wear is much higher at the front then the back wheel, the force distribution is modelled using a trapezoidal distribution, Figure 2.

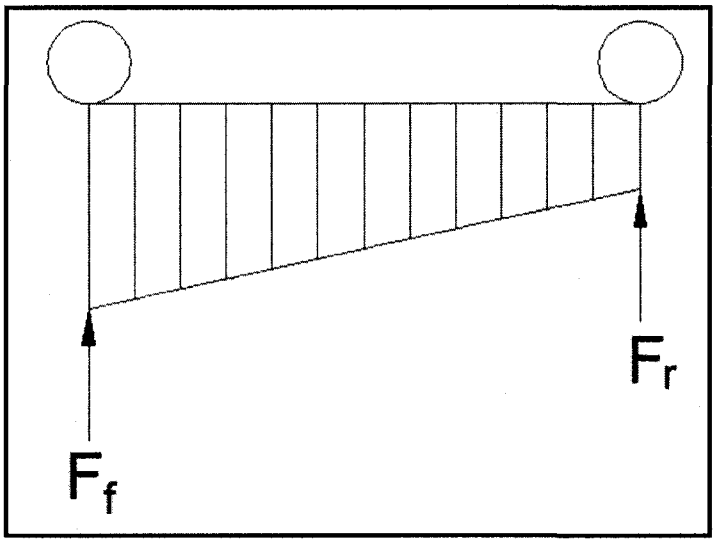


Figure 2: Force distribution between the front and rear wheel of the bulldozer

In order to calculate F_f and F_r, we need to find the torque generated by the overburden being pushed in front of the blade. This torque is equal to the force the overburden is exerting on the blade multiplied by its height. For different volumes of overburden the bulldozer is pushing, corresponding height of the overburden are determined, this is summarized in table 1.

Table 1: % of Blade volume versus height of overburden

Blade Volume (%)	Height of overburden
100	0.968
90	0.919
82	0.881
78	0.886
74	0.889
70	0.894
65	0.899
58	0.909
50	0.922
43	0.937
34	0.871
17	0.711
13	0.693

By determining the volume of overburden the bulldozer is pushing, we can determine the height, and we can calculate the force distribution across the wheels of the bulldozer. The emphasis of this research is to create a model that not only allows us to predict the volume of overburden the bulldozer is pushing as a percentage of blade volume, but also to determine the factors that influence the volume of overburden being pushed by the bulldozer.

Creating a model that will allow us to predict the volume of material pushed by the bulldozer, allows us to construct a system that can equalize the force distribution across the wheels making them wear a lot less and at equal time intervals reducing maintenance costs, and improving operations, and downtime.

1.4 Contributions

At the beginning we should emphasise the fact that this kind of research is very innovative, and we could not find any indication of similar research activities. Consequently, there is no literature dealing with this subject matter.
The contributions of the thesis are following:
collection of on-site measurements of heavy earth-moving equipment, performed during different times during a year – in particular in January and March; this task has been done with a tremendous of help of a dedicated individual from an oil sand company;
processing of the measurement data sets; this task has been preformed in order to ensure consistency of data points; the validation of data has been accomplished also with a help of the individual from the oil send company;
development of neuro-fuzzy models of the measurement data; this task has been preformed using an evolutionary-based optimization system for model construction.
Overall, the research activities have resulted in construction of a system for predicting the volume of material pushed by the bulldozer. Analysis of the constructed model (system) allows for exploring the factors that influence relationships between the volume and measures quantities.

1.5 Thesis outline

This thesis is divided into 7 chapters. The first chapter is an introduction, and problem description. Chapter 2 introduces the methods used in this research, and gives a brief introduction to each of them. Chapter 3 describes the prediction system proposed in and deals with the method used in our model, and its learning mechanism, chapter 4 describes the data used in our research, chapter 5 describes details regarding the structure of the system. Chapter 6 deals with the results and models obtained from experiments, describes the knowledge gained from the system. Chapter 7 contains the conclusion, and future recommendations.

Chapter 2

Background

The field of Computational Intelligence and Artificial Intelligence in general is concerned with mimicking the information processing capabilities of the human brain, in order to solve a variety of engineering problems. The human brain operates on two basic levels, the physical level, which involves the processing nodes, called neurons, and the logical level, which involves the decision making process, and dealing with uncertainty, as well as incomplete information.

The physical level of the human brain involves small processing units called neurons. Each neuron consists of the dendrites, which receive inputs from the neurons, the soma or cell body, which supports the functionality of the neuron, and processes the incoming signals, the axon, which transports the resulting signal, and synapse, which contains the neurotransmitters that propagate the signal to the next neuron.

There are approximately 100 billion neurons in the human brain, each of which is connected on average to thousands of other neurons. The response time of a single neuron is about 1 millisecond, orders of magnitude less than today's powerful processors, which have processing times in the nanoseconds. However, what the brain lacks in speed is made up for in parallel processing capability, instead of one processor working, we have hundreds of thousands or even millions of neurons working together to achieve a common task.

The logical level of the brain is the brains ability to deal with incomplete or uncertain information, and arrive at the correct conclusion or course of action to take, and process

language. Making use of this logical processing capability of the brain allows us to construct systems that can process and understand natural language, and process information in the presence of uncertainty and incompleteness.

2.1 Neural networks

Neural networks were first introduced by McCulloch and Pitts in 1943, they have introduced a simple and basic construction of an artificial neuron, called the "McCulloch Pitts neuron" Figure 3. This early model of the biological neuron, have demonstrated abilities to solve the logical binary operations.

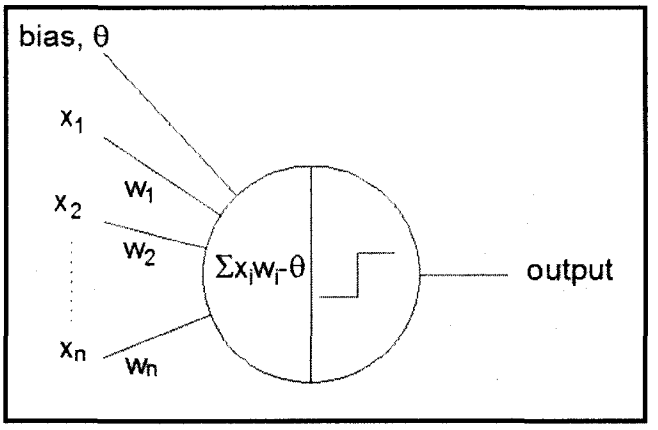


Figure 3: The McCulloch Pitts neuron

The inputs to the neuron $x_1, x_2, ..., x_n$ are multiplied by their corresponding weights $w_1, w_2, ..., w_n$, which represent the synaptic strength between biological neurons. The net input to the neuron is the sum of the product of the inputs and their corresponding weights minus a bias or threshold term.

$$neuron_{in} = \sum_{i=0}^{n} w_i x_i - \theta$$

This input then undergoes a transformation using a stepwise transfer function $f(neuron_{in})$ to produce the final output, if the final output exceeds the bias or threshold θ, the neuron fires. The transfer function used in the McCulloch Pitts neuron is a binary output function

$$f(neuron_{in}) = \begin{cases} 1 \; if \; neuron_{in} \geq \theta \\ 0 \; if \; neuron_{in} < \theta \end{cases}$$, where θ is the bias or threshold of the neuron

The McCulloch Pitts neuron had disadvantages, which have limited its application to binary tasks:

1) No mechanism for learning the weights, and threshold of the neuron
2) The output of the neuron is binary, not suited for regression tasks

In 1949 Donald Hebb proposed a mechanism of learning in his book "The Organization of Behavior". The method became known as Hebbian learning:

"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased"[1]

In 1958, Rosenblatt introduced the perceptron, in which he integrated all the previous work of McCulloch, Pitts and Hebb. He developed a model of the neuron that is capable of learning from sample of input output data points, by adjusting the weights of the connections between neurons.

The learning rule is called the "perceptron learning rule". Given an input output pair of training points (x_i, y_i), all weights are initialized to random values, usually in the range [-0.5, 0.5] and compute the output of the neuron t_i. If the desired value y_i equals the neuron output t_i, then there is no need to change weights, otherwise the weights are adjusted according to:

$w_{new} = w_{old} + \eta * (y_i - t_i) * x_i$, where η is the learning rate

The learning rate η is a constant between 0 and 1 that determines how fast the perceptron learns the new weights. Setting this value too high, will cause the weights to oscillate, and setting it too low, the network will take long to learn. This learning process is repeated until the weights of the network do not change any more, and the network is said to have converged.

The transfer functions the perceptron uses can be any of a number of functions, some of the transfer functions are shown in Figure 4.

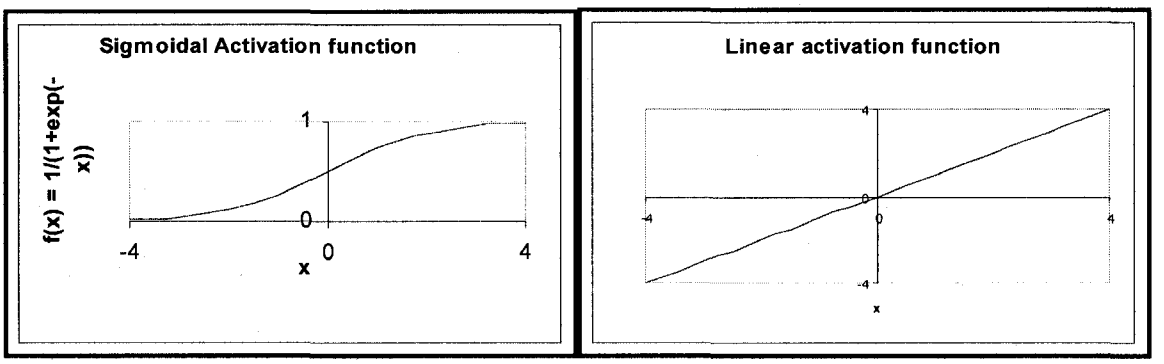


Figure 4: Examples of transfer functions

Later, in 1969 Minsky and Papert showed that there are some limitations of perceptrons, It is not able to solve for example, the XOR problem, and that a network of perceptrons arranged in layers are better capable of solving these problems, this network is called multilayer perceptron.

2.1.1 Types of Neural Networks

Neural networks fall under different types, according to their structure or learning mechanism. Multilayer perceptrons are one kind of neural network in which the inputs are propagated forward through the layers also called (Feed forward neural network), this is by far the most popular type of neural networks, Figure 5. This type of network is most suitable for classification and regression problems.

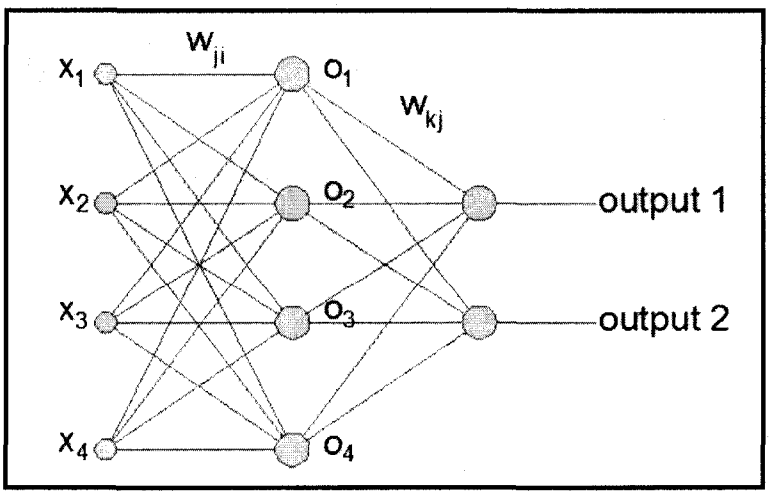


Figure 5: Multilayer perceptron

Radial basis functions are another type of neural networks that have the same structure as the multilayer perceptron. However, the neurons in the hidden layer consist of basis functions instead of the sigmoid activation functions, and the maximum number of hidden layer neurons equals the number of inputs. The output of the neural network is the weighted sum of the basis function outputs, and there are no weights in the input layer. This particular type of neural network is highly suitable for function approximation and regression problems.

Another type of neural networks is the recurrent neural network. This type of network is capable of remembering its previous state, because there is a feedback from layers or neurons. There are two kinds of feedback, global and local feed backs. In global feedbacks, a neuron in the next layer feeds a neuron in the previous layer, or a neuron in the same layer with its output, and in local feedback, a neuron feeds its own with its output. Recurrent networks are most suitable for storing and retrieving information, the Hopfield network, Figure 6, is a global feedback type recurrent neural network, which mimics the memory centers in our brain.

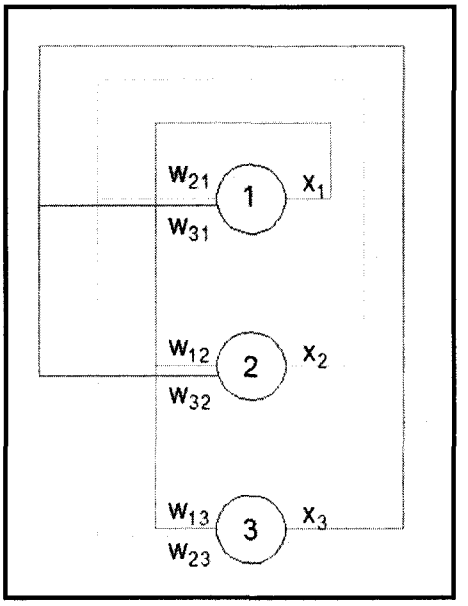


Figure 6: The Hopfield network

Another type of neural networks is the Kohonen self organizing maps (SOM). In this kind of network, the inputs are fully connected to the output neurons, and each input vector is compared with the weight from the input vector to the output neuron, the output neuron which is closest to the input vector is the winner, weights are updated, and process repeated until the termination criteria is met. Figure 7 shows a Kohonen self organizing map. Kohonen self organizing map is most suitable for pattern recognition and clustering problems.

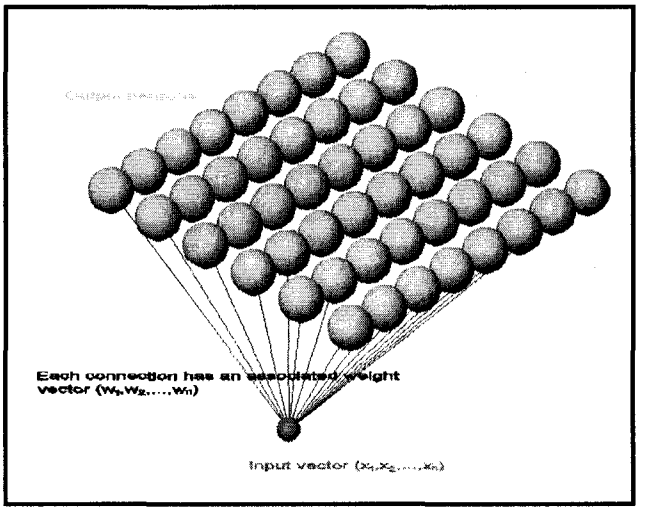


Figure 7: Kohonen self organizing map

2.1.2 Learning Techniques

The strength of neural networks arises from their ability to learn, and adapt to new inputs. This is achieved by adjusting the weights of connections between the neurons, or the synaptic strengths. Different neural networks require different learning mechanisms, however the common theme is adjusting the weights is how neural networks in general learn. The exception to this rule is the Hopfield network.

There are three kinds of learning methods:

1) Supervised learning
2) Unsupervised learning
3) Reinforcement learning

Each of these learning methods is used to train different structures. In supervised learning techniques, an input, output pairs of training points are provided. The inputs are propagated through the network and learning takes place because the network compares its output with the desired output, and adjusts its weights accordingly. This type of learning is used to train the multilayer perceptron, and radial basis function network.

Training the multilayer perceptron requires us to find a way of propagating the weight changes in the output layer to the hidden and input layers. This is done by the back-propagation algorithm, which was invented by Werbos in 1974 and re-invented again by Rumelhart and McClelland in 1986. The algorithm uses gradient decent minimization to find the weights which will result in the lowest error.

Gradient decent is given by:

$$\Delta w_{ij} = -\eta \nabla E(w_{ij}) = -\eta \frac{\partial E(w_{ij})}{\partial w_{ij}}$$

We change the weights of the network, in the opposite direction of the gradient, so as to minimize the error. The total error given after applying the inputs to the neural network is

$$E = \frac{1}{2} \sum_{p=1}^{p=n} \sum_{j=1}^{j=m} (t_j - o_j)^2,$$

where

n is the number of training patterns
m is the number of outputs
t_j *is the desired output of the neural network*
o_j *is the actual output of the neural network*
E is the total error

Two modes of learning are possible with neural networks, online and batch training. In online training, the weights are updated every input pattern, and in batch training, the weights are updated every complete run of the patterns through the network, which is

called epoch. The difference in both is a drop of the summation over all patterns. Since the output of the neuron is a function of its net inputs,

$$net_j = \sum_{i=1}^{i=k} w_{ij} x_i$$

$$o_j = f(net_j)$$

Using online training, the error as a function of the neuron input becomes,

$$E = \frac{1}{2} \sum_{j=1}^{j=m} (t_j - f(net_j))^2$$

Calculating the gradient of the error

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = 2 \times \frac{1}{2} \times \sum_{j=1}^{j=m} (t_j - f(net_j)) \times f'(net_j) \times (-x_i)$$

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = \sum_{j=1}^{j=m} (t_j - o_j) \times f'(net_j) \times (-x_i)$$

For the sigmoidal transfer function, $f'(net_j) = o_j \times (1 - o_j)$, and dropping the summation sign,

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = (t_j - o_j) \times o_j \times (1 - o_j) \times (-x_i)$$

The term $\delta = (t_j - o_j) \times o_j \times (1 - o_j)$ is called the error signal, and x_i is the ith input to the j^{th} neuron.

The change in weight for the output layer becomes,

$$\Delta w_{ij} = -\eta \frac{\partial E(w_{ij})}{\partial w_{ij}} = -\eta \times \delta_j \times (-x_i) = \eta \times \delta_j \times x_i$$

Since we don't have a desired output to compare the output of the hidden neurons to, we back propagate the output layer error into the hidden nodes, the error signal for the hidden neurons becomes $\delta_h = o_h \times (1 - o_h) \times \sum_{j=1}^{j=m} (w_{hj} \delta_j)$, the weights of the hidden neurons are then updated using $\Delta w_{kh} = \eta \times \delta_h \times y_k$, were y_k is the input to the hidden node.

The initial weights of the Hopfield network on the other hand are set using $w_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j$, there is no feedback between the neuron and itself, consequently the weight between the neuron and itself is zero. With the introduction of an unknown state, the network will converge to the closest remembered pattern. Hopfield defined the energy of a state to be $e = w_{ij} x_i x_j$, the remembered patterns will have the lowest energy, and an unknown pattern will converge to the lowest energy directly, or through a number of state transitions. The state of the network is updated either synchronously (update the output of one neuron each time step) or asynchronously (update all the outputs of all neurons each time step).

The radial basis function requires first, determining the cluster centers and spread of each of the basis functions, and training the output weights of the network. Training the output weights is done using gradient decent. However, finding the center and spread of the basis functions, can be done using unsupervised learning techniques.

In unsupervised learning, the network is provided only with the inputs, and it's up to the network and the learning algorithm to be able to find structures within the data. There is no guidance as to what the outcome should be. The Kohonen self organizing map is an example of unsupervised learning method. The weights of the connections are initialized, and each input pattern is compared to all weight vectors. The closest one to the input pattern is the winner, and then the weights are updated via

$w_{t+1} = w_t + h_t \times [x_t - w_t]$, where
w_{t+1} is the updated weight vector
w_t is the previous weight vector
h_t is a neighbourhood function
x_t is the input vector

In reinforcement training, the network is only presented with feedback on whether the network is doing well or not, there is no training data that the network can compare its outputs to, only feedback on the actions of the network, for example, touching the fire is bad "you burn your finger and feel pain", the result is "don't touch fire".

2.2 *Fuzzy Systems*

In everyday life, we use terms like tall, short, big, small, etc, to describe different things. These terms are not precisely defined, and often values can belong to two or more different sets. In classical sets, an item either belongs to a set or does not Figure 8, there is no tolerance for imprecision or vagueness.

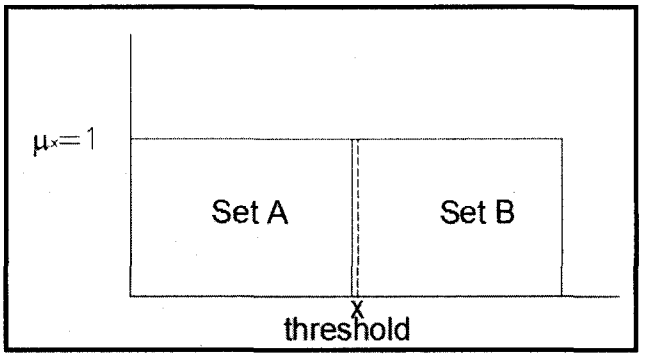


Figure 8: Crisp set

A famous paradox that illustrates the deficiencies of classical sets in real life is the "sorites paradox", which states the following two premises:

1,000,000 grains of sand is a heap of sand. (Premise 1)
A heap of sand minus one grain is still a heap. (Premise 2) [2]

If we were to continually repeat the second premise, at some point, a heap will consist of a single sand grain, which we know is incorrect.

This paradox can be applied to all linguistic terms, which shows that uncertainty and vagueness are part of our everyday life. In 1965, Lotfi Zadeh, introduced the concept of fuzzy logic, which allows objects to belong partially to a number of sets. With the introduction of fuzzy logic, we are able to model real world events more realistically.

2.2.1 Basic concepts

Fuzzy logic assigns a membership value $\mu(x)$ to each element in each set, so that an element could belong to two ore more sets in the same time to different degrees, Figure 9.

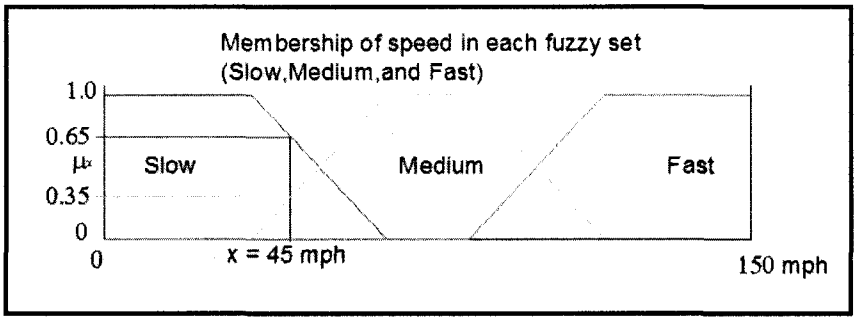


Figure 9: Fuzzy sets

In Figure 9, a speed of 45 mph has membership in the slow set of 0.65, the medium set of 0.35, and to the fast set to of 0. The membership is not a probability it is only a degree of belonging to a specific set. The membership functions in Figure 7, which describe the degree each of the inputs belongs to the fuzzy set is trapezoidal in nature.

Each of the fuzzy sets in Figure 7 can be written as:

$\mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + ... + \mu_A(x_n)/x_n$, where

x_i is the crisp values for which the memberhip function is defined

$\mu_A(x_i)$ is the membership of x_i in fuzzy set A

The universe in which the x_i's are defined is called the universe of discourse.

The number of membership functions, and membership function shapes are dependant upon the problem to be solved. There are a variety of membership functions, trapezoidal, triangular, and Gaussian membership functions.

Operations on fuzzy sets include union, intersection, and complement. The intersection of two fuzzy sets, A and B is defined as

$A \cap B = \min(\mu_A(x), \mu_B(x))$

The union of two fuzzy sets A and B is defined as

$A \cup B = \max(\mu_A(x), \mu_B(x))$

And the complement of a fuzzy set A is defined as

$\mu_{\bar{A}} = 1 - \mu_A(x)$

A more general form of the union and intersection operators is the T and S norms.

The T-norm is mapping T: $[0,1]\times[0,1] \rightarrow [0,1]$, which satisfies the following:

Commutativity : $T(a,b) = T(b,a)$
Monotonicity : $T(a,b) \leq T(c,d)$ if $a \leq c$ and $b \leq d$
Associativity : $T(a,T(b,c)) = T(T(a,b),c)$ [4]
$T(a,1) = a$

The S-norm is mapping S: $[0,1]\times[0,1] \rightarrow [0,1]$, which satisfies the following:

Commutativity : $S(a,b) = S(b,a)$
Monotonicity : $S(a,b) \leq S(c,d)$ if $a \leq c$ and $b \leq d$
Associativity : $S(a,S(b,c)) = S(S(a,b),c)$ [5]
$S(a,0) = a$

2.2.2 Fuzzy if then rules

Fuzzy if then rules are of the form:

If (antecedent) Then (consequent)

The antecedent is formed by the conjunction or disjunction of the fuzzy terms. An example of a fuzzy rule is:

If "the temperature is hot or humidity is high" Then "fan speed is high"

Fuzzy rules require an inference mechanism to be able to be useful. For the conjunction or disjunction of the antecedents, we use the T or S norm, which corresponds to the intersection or union of the fuzzy sets. The value of conjunction or disjunction of all these fuzzy terms, will give us the degree of satisfaction of the consequent, which refers to the degree to which the consequent is true.

Now, we need to propagate the degree to which the consequent is satisfied to the consequents fuzzy set, which is called implication. There are a variety of implications available, some of which are:

Larson implication

$$\mu_{A\to B}(x,y) = \mu_A(x) \times \mu_B(y)$$

Mamdani implication

$$\mu_{A\to B}(x,y) = \min[\mu_A(x), \mu_B(y)]$$

Zadeh implication

$$\mu_{A\to B}(x,y) = \max[\min[\mu_A(x), \mu_B(y)], 1 - \mu_A(x)]$$

When multiple rules exist, we need to aggregate the output of each rule. The aggregation operator is usually the maximum. After we aggregate the rules, we have the final output membership, which can be defuzzified to produce a crisp output. There exist many defuzzification methods, some of which are:

Center of Area, Center of gravity defuzzification

$$COG = \frac{\sum_{x=\min}^{x=\max} x \times \mu(x)}{\sum \mu(x)}$$

Mean of maxima defuzzification

$$MOM = \frac{\sum x}{\max \mu_x}, \text{ where}$$

n is the number of values for which μ_x is maximum

Fuzzy mean defuzzification

$$FM = \frac{\sum_{i=1}^{n} \alpha_i c_i}{\sum_{i=1}^{n} \alpha_i}, \text{ where}$$

n is the number of fuzzy output sets

c_i is a characteristic value of the output fuzzy sets, *COG* or *MOM*

This crisp output will be the output of the fuzzy system.

2.3 Neurofuzzy Systems

Neurofuzzy systems combined the benefits of both fuzzy systems, and neural networks. Neural networks have the ability to learn and generalize, while fuzzy systems are good at accounting for uncertainty, and comprehensibility, since you can see and the relationship between variables. Neurofuzzy systems combine the benefits of both to create a system that is capable of both learning, tolerance of uncertainty, and transparency.

2.3.1 Architectures

Neurofuzzy systems are of three types [9]:

1) Cooperative neurofuzzy network, Figure 10.
2) Concurrent neurofuzzy network, Figure 11.
3) Hybrid neurofuzzy networks

In cooperative neurofuzzy networks, the neural network adapts the fuzzy system. This included training the weights for the fuzzy rules, or tuning the membership functions of the fuzzy system. After training the weights, or tuning the membership functions of the fuzzy system, the neural network is disconnected, and the fuzzy system operates without the need for the neural network.

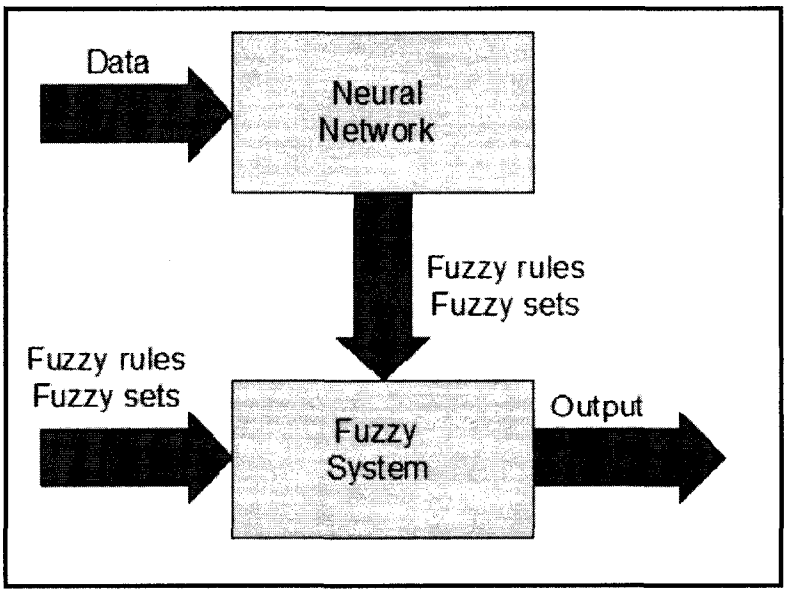


Figure 10: Cooperative neurofuzzy network

In concurrent neurofuzzy systems, a neural network and the fuzzy system are connected so that the output of one is fed into the other. The neural network does not actually modify the fuzzy system; however, it modifies the input or output of the fuzzy system as a pre-processing or post-processing step.

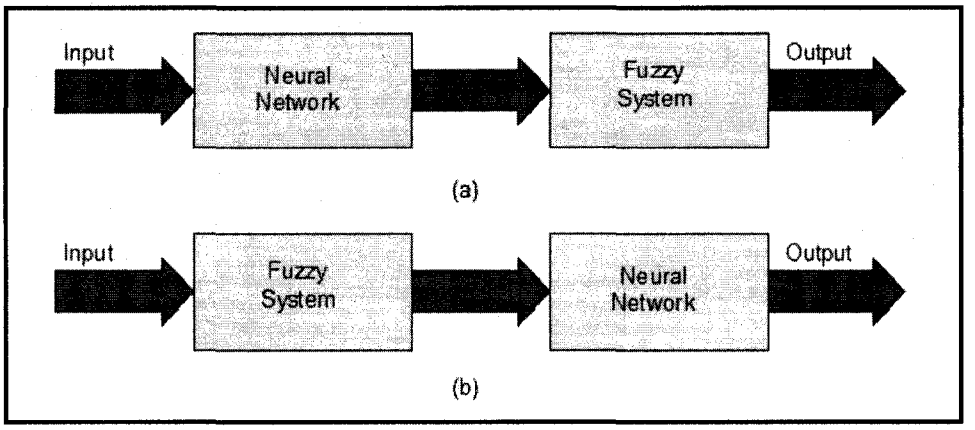


Figure 11: Concurrent neurofuzzy networks

Hybrid neurofuzzy system, incorporate the principles of fuzzy logic, into neural networks. The neural network is modified so that the neurons are fuzzy neurons [6], and the inputs are fuzzy inputs. The fuzzy neurons are the AND, OR fuzzy neurons Figure 12.

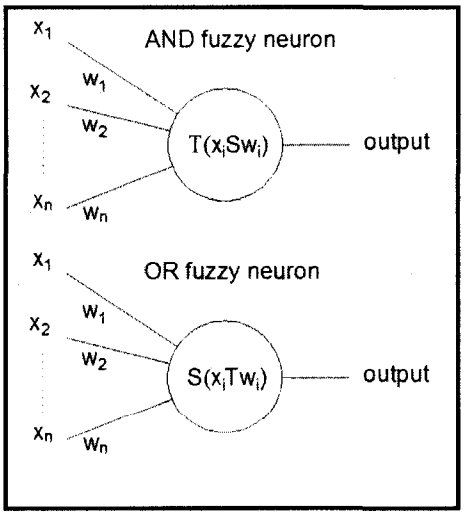


Figure 12: Fuzzy AND, OR neuron.

T and S norms are used instead of the sigmoidal activation function.

$$AND = T(S(x_1, w_1), S(x_2, w_2), ..., S(x_n, w_n))$$

$$OR = S(T(x_1, w_1), T(x_2, w_2), ..., T(x_n, w_n))$$

In hybrid neurofuzzy systems, the input layer consists of fuzzification of the raw inputs, followed by the rules layer, and the output layer. There are three inference system types used in fuzzy and neurofuzzy systems.

1) Mamdani based inference system
2) Sugeno-Takagi based inference system
3) Tsukamoto fuzzy inference system

In the Mamdani based inference system, the inputs to the fuzzy system are fuzzified and then send to the corresponding rules. The outputs of the rules are then sent to their respective consequents, combined, and the output is defuzzified.

In the Sugeno-Takagi based inference system, the output from each fuzzy rule takes the form

If x *is* A and y *is* B *Then* $C = f(x, y)$, where

$f(x, y) = px + qy + r$

For a first order system, the parameters p, q, r are learned parameters. In a Sugeno-Takagi neurofuzzy system, the outputs from each rule are normalized in a normalization layer to determine the firing strength of each rule, given by

$\overline{\alpha}_i = \dfrac{\alpha_i}{\sum_{i=1}^{i=n} \alpha_i}$, where

$\overline{\alpha}_i$ is the normalized firing strength of the rule

α_i is the firing strength of the rule

The output of the fuzzy system is then taken to be

$z = \sum_{i=1}^{n} \overline{\alpha}_i f_i(x, y)$, where

n is the number of rules

In the Tsukamoto fuzzy inference system, the consequents membership functions are monotonic, and the resulting system output is the weighted average of the rules.

2.3.2 *Learning techniques*

In both the concurrent and cooperative neurofuzzy system, the learning techniques applicable to neural network training are used, which mainly is the backpropagation learning algorithm. For the hybrid neurofuzzy system, the backpropagation learning technique also applies, however it requires some modification, since we have fuzzy neurons instead of our normal neurons.

$$\Delta w_{ij} = -\eta \nabla E(w_{ij}) = -\eta \frac{\partial E(w_{ij})}{\partial w_{ij}}$$

For the output layer, we have the OR neuron, and using online learning,

$$E = \frac{1}{2} \times (t_j - o_j)^2$$

t_j is the desired output of the neural network

o_j is the actual output of the neural network

E is the total error

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \times 2 \times (t_j - o_j) \times \frac{\partial o_j}{\partial w_i} = (t_j - o_j) \times \frac{\partial o_j}{\partial w_i}$$

The output of the OR neuron is given by:

$o_j = \underset{i=1}{\overset{n}{S}}(w_i T x_i)$, where

S is the S - norm

T is the T - norm

Using the product T-norm, and probabilistic sum S-norm, we can write the output of the OR neuron as:

$$o_j = 1 - \prod_{i=1}^{n} (1 - x_i w_i)$$

The derivative of the output with respect to the weight becomes,

$\frac{\partial o_j}{\partial w_i} = x_i \prod_{\substack{j=1 \\ i \neq j}}^{n} (1 - x_j w_j)$, which can be written as

$$\frac{\partial o_j}{\partial w_i} = x_i \times (1 - \underset{\substack{j=1 \\ i \neq j}}{\overset{n}{S}} (w_j T x_j))$$

The error signal of the output is

$$\delta_j = (t_j - o_j) \times (1 - \underset{\substack{j=1 \\ i \neq j}}{\overset{n}{S}} (w_j T x_j))$$

$$\frac{\partial E}{\partial w_i} = (t_j - o_j) \times (1 - \underset{\substack{j=1 \\ i \neq j}}{\overset{n}{S}} (w_j T x_j)) \times x_i$$

To backpropagate the error, we need to calculate the derivate of the AND, with respect to the input weights. The output of the AND neuron is given by:

$o_h = \underset{m=1}{\overset{k}{T}} (w_m S x_m)$, where

S is the S - norm

T is the T - norm

The output of the AND neuron can be written as:

$$o_h = \prod_{m=1}^{k} (x_m + w_m - x_m w_m)$$

The derivative of the output with respect to the weight becomes,

$\frac{\partial o_h}{\partial w_m} = (1 - x_m) \times \prod_{\substack{l=1 \\ m \neq l}}^{k} (x_l + w_l - x_l w_l)$, which can be written as

$\frac{\partial o_h}{\partial w_m} = (1 - x_m) \times (\underset{\substack{l=1 \\ m \neq l}}{\overset{k}{T}} (w_l S x_l))$, the error signal for the input layer is

$$\delta_h = (1 - x_m) \times (\underset{\substack{l=1 \\ m \neq l}}{\overset{k}{T}} (w_l S x_l)) \times w_{hj} \times (t_j - o_j) \times (1 - \underset{\substack{j=1 \\ i \neq j}}{\overset{n}{S}} (w_j T x_j)$$

2.4 Evolutionary computation

In 1859, Charles Darwin published his book titled "Origin of Species", in which he introduced his theory of evolution. Darwin argued that species evolve through a process called natural selection, where the fittest individuals in the population survive. The fittest

individuals are the ones which are better adapted to live in their environment. Occasionally mutations occur in the population, and if the mutation leads to a better suited individual, then that individual thrives. Reproduction also might generate better suited individuals, over time the population would be highly suited to its environment.

In 1975, Holland introduced genetic algorithms, which are based on Darwin's theory of evolution. Holland used the concepts of natural selection, mutation, and reproduction to find solutions to optimization problems.

2.4.1 Concepts – Genetic Algorithm

Genetic algorithms attempt to mimic the natural process of evolution to solving optimization and search problems. Initially optimization space of the problem is identified, a population of possible solutions are generated, and encoded (called chromosomes), and their fitness evaluated using a fitness function, which measures how good the individuals in the population are. The selection process is carried out to select the best individuals in the population. There are multiple selection mechanisms, some of which is the roulette wheel selection, and rank selection.

Some of the individuals in this population randomly undergo mutation and reproduction (also called crossover), generating new individuals or offspring, the individuals that don't are copied to the next generation. The population size is kept constant through out the whole process. After the process is done, we have a new population of individuals, with higher overall fitness. This process is repeated for a number of generations until the stopping criterion is met. Figure 13 is a flow diagram of a genetic algorithm.

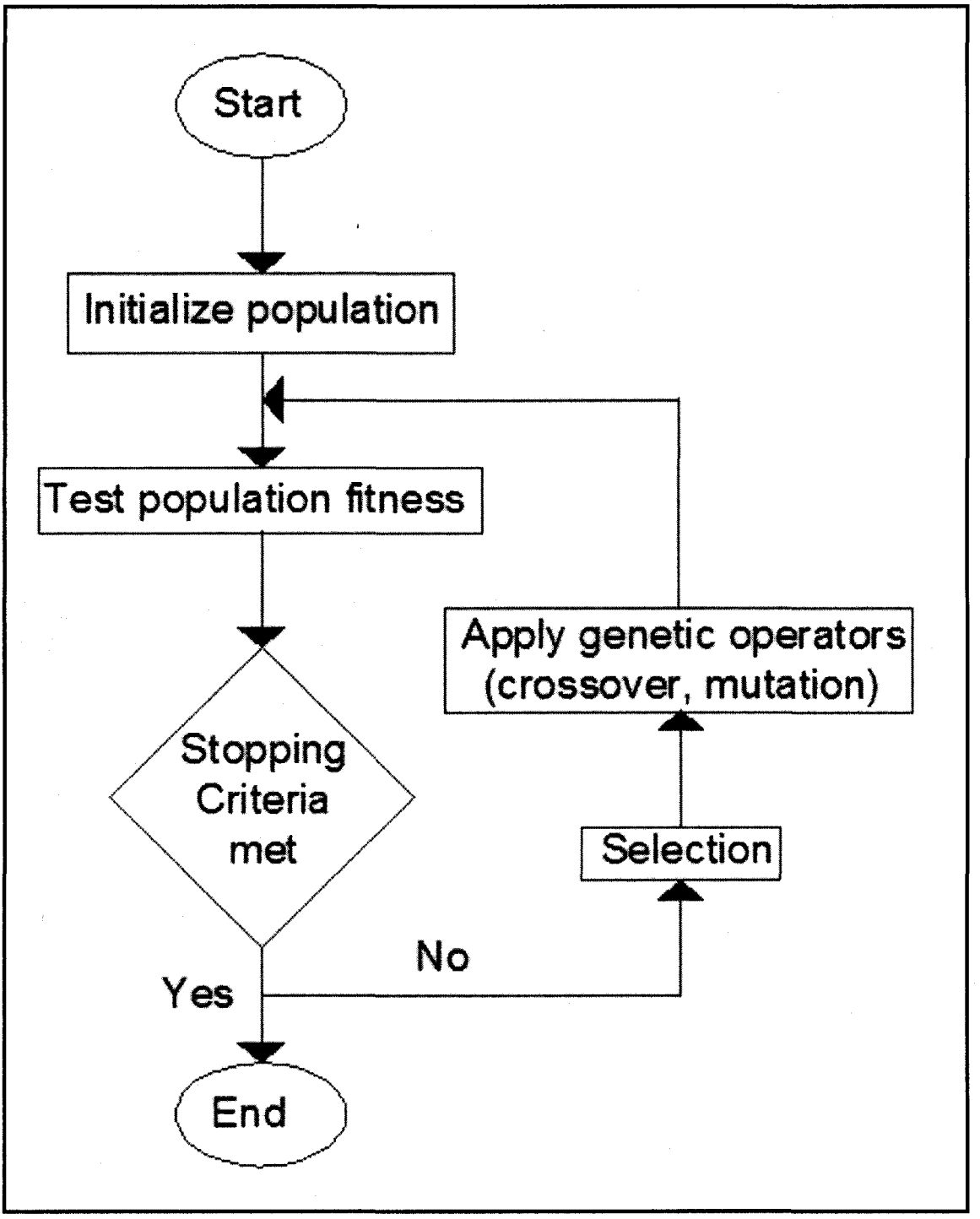


Figure 13: Genetic algorithm flow diagram

Genetic algorithms require choosing an encoding scheme, to encode the individuals in the population, determining and setting some parameters, and the determination of the fitness function, which dictates the solution space that is to be searched.

Genetic algorithms typically encode the solutions in binary format. Gray coding or other encoding scheme can be used.

The mutation operator allows for the exploration of different areas in the solution space, setting the rate of mutation dictates the amount of individuals in the population that will undergo mutation. If this percent is too high, the individuals within the population will mutate very frequently, and a good solution could be lost, on the other hand, if the rate of mutation is too low, we are not exploring the solution space enough. There is no particular rule that we can use to set the mutation rate, so it's a matter of experimentation.

The reproduction operator allows us to explore the region around a particular individual in the population, it is a local search. The crossover rate, dictates the amount of individuals that will undergo crossover. If this rate is too high, the genetic algorithm will not be able to explore other regions in the solution space, and if this rate is too low, the genetic algorithm will not have a chance to explain the region surrounding the individuals in the population. There is no rule for selecting the crossover rate.

The number of generations determines the evolution time, the larger, the better, since the genetic algorithm will explore more of the solution space. Population size determines the number of individuals in our population, the larger the size, the better, since we have more points to explore in the solution space.

The fitness function is the heuristic that guides the evolution process. Selecting the correct fitness function for the genetic algorithm is essential to obtaining good results. If the fitness function is not representative of the problem to be optimized, then an optimal solution may not be found.

Genetic algorithms are not guaranteed to find the best solution. Also there is the possibility that the genetic algorithm might get stuck at a local maxima, this problem is

more apparent with functions that are multimodal, where the fitness function contains a lots of peaks.

2.4.2 Genetic Programming

Genetic programming was pioneered by John Koza for optimization of complex problems. Genetic programming uses, and works by the same concepts and procedures as genetic algorithms, however, in genetic programming the solutions space is not composed of chromosomes, but of programs or solutions represented by a tree structure, Figure 14.

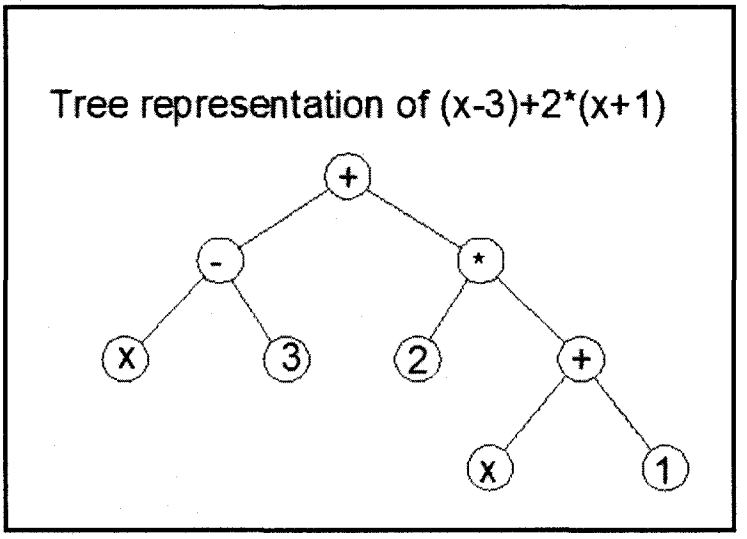


Figure 14: A program and its tree representation

To prepare the GP for solving a problem, we first must define the following:

1) Function set
2) Terminal set
3) fitness function
4) mutation rate
5) crossover rate
6) number of generations
7) populations size

The function set is the operations that the solution should be composed of, like +,-,/,*,sqrt,..,etc, and the terminal set is the operands, or the variables and constants that the

solution should include. Defining the parameters and the fitness function is the same as that of the genetic algorithm.

The steps that the GP follows are identical to that of the genetic algorithm. Crossover in GP is carried over by cutting two branches of two individuals and exchanging them, Figure 15.

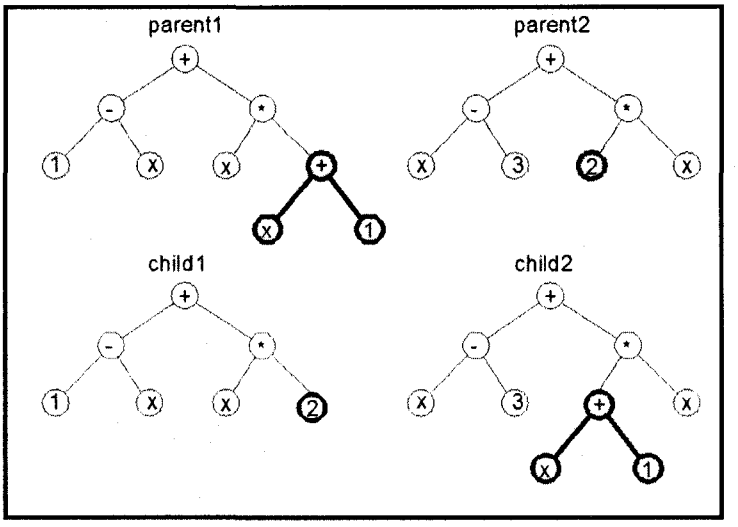


Figure 15: Crossover operation in GP

Mutation in GP involves cutting down a branch of the tree, and replacing the branch with a newly generated one, Figure 16.

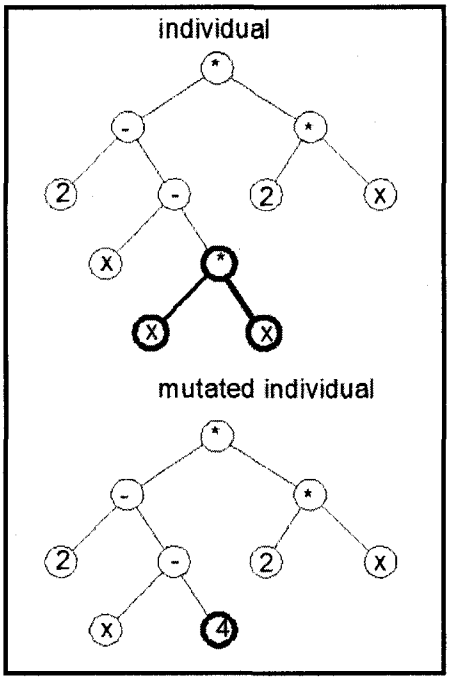


Figure 16: Mutation in GP

Care must be taken, when doing crossover and mutation, as we require the mutated individual or children to be a valid solution. GP is subject to the same problems that might affect the genetic algorithm, including being stuck on local maxima.

There are three methods for creating the tree programs:

1) full method
2) grow method
3) ramped half and half

In the full method, the individuals in the population are created down to the maximum depth, D of the tree. In the grow method, the individuals are created anywhere up to the maximum depth D of the tree, this method allow for greater variation in the population, then the full method. The ramped half and half method is a combination of both, were half of the population is created using the full method, and the other half is created using the grow method, this method provides for the greatest variation in the population.

Chapter 3

Neurofuzzy Prediction System

3.1 Concept

The technique we have chosen to tackle our problem is using a hybrid neurofuzzy system for prediction, for three main reasons:

The system is capable of learning and generalizing from training data
The system is capable of handling uncertainty
the system is transparent, we can see and understand the relationship between variables

Neural networks are adaptive systems that learn from the inputs, and when properly trained, they generalize well from data. The fact that our research data contain uncertainties in the values of the volume measured means that we require a system that is capable of incorporating this uncertainty as well as being transparent, so we can determine the relationships between all the variables. A hybrid system that combines the previously mentioned benefits of neural networks with fuzzy systems is a good choice for a modeling the forces acting on the bulldozer.

3.2 Architecture

Figure 17 is a simplified diagram of a hybrid neurofuzzy system, the inputs are fuzzified using their membership functions, and the fuzzified inputs are send to neurofuzzy network, the network produces the fuzzy outputs which are defuzzified to crisp values.

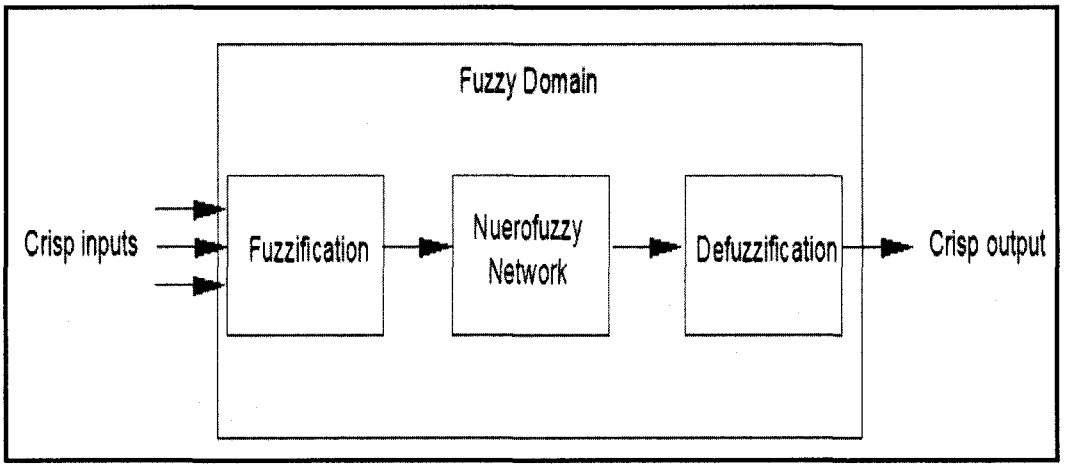
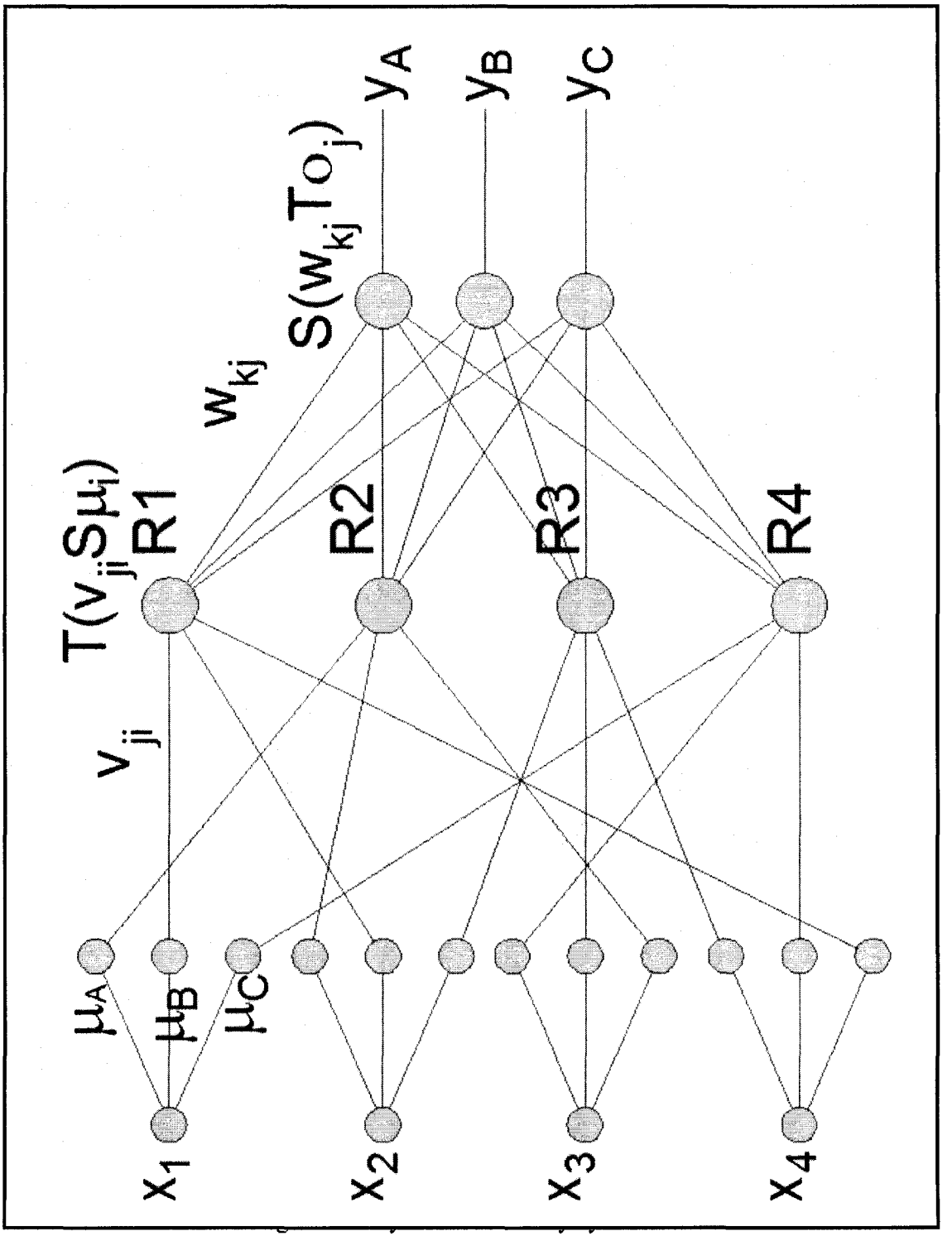


Figure 17: Simplified hybrid neurofuzzy network

The system consists of the following layers:

1) The input layer
2) The fuzzification layer
3) The rules layer
4) The output layer

Figure 18 is a diagram of the hybrid neurofuzzy system. The inputs to the neurofuzzy system are fuzzified, and the fuzzified inputs are send to their corresponding rules, there T norms are calculated. The output from the rules layer is send to the corresponding consequents membership function, and aggregated to produce the final degree of satisfaction of the consequents output fuzzy set.



The system is based on the Mamdani inference; we produce the degree of satisfaction of the consequents fuzzy sets, which we then use the fuzzy mean defuzzification described in chapter 2 to calculate the final crisp output.

3.3 Construction

In the input layer, the inputs are presented to the system, and each input is fuzzified using Gaussian membership functions of the form

$$\mu_i(x) = e^{\frac{-(x-\mu)^2}{\sigma^2}}$$

The mean and standard deviation of the Gaussian membership functions are determined using

$$\mu_i = \min_i + \sum_{i=0}^{n-1} \frac{(\max_i - \min_i)}{(n-1)} \times i, \text{ where}$$

n is the number of membership functions

$$\sigma_i = \frac{0.45 \times \frac{(\max_i - \min_i)}{(n-1)}}{-1.0 \times \log(0.45)}$$

The means of the membership functions are determined so that we have equal intervals between the means of the membership functions, and the standard deviation is the same for all membership function.

After the fuzzification stage, the selected fuzzy inputs are fed into the rules layer, which consist of the AND neurons. The T-norm is the product of the inputs $T(x, y) = xy$. The S-norm is the probabilistic sum $S(x, y) = x + y - xy$. These T and S norms were chosen because we need to have differentiable functions, so that we can use the backpropagation learning algorithm.

The outputs from the rules are then fed into the output layer, which consist of the OR neurons. Each output from each neuron is fed into its corresponding output neuron, and all the outputs are combined together to yield a degree of satisfaction of the consequents

fuzzy set. The neurofuzzy system is trained using the backprogation learning algorithm as described in chapter 2.

The initial weights of the inputs are 0 for the inputs to the AND neuron, and 1 for non inputs to that AND, For the OR neuron, the weights get reversed, so that the weights become 0 for non inputs to the OR, and 1 for inputs. This is required because of the properties of T and S norms. When these weights get trained using the backpropagation, only the weights that correspond to the actual inputs are modified.

Because of the rule explosion, we have determined that the best membership functions are 3 for both inputs and outputs, if we were to use a bigger value of the membership functions, the amount of potential rules will be very high, since we have 8 inputs to the network, this will also give us a reasonable view of the data, without having too much terms, and making interpretation difficult, and too few terms which are not sufficient to describe the data with good detail.

Determining the inputs to each rules, and the corresponding output are a knowledge that have to be determined a priori, or via validation runs, to pick the structure which yields the best results. Because of the rule explosion, in which the number of rules grows exponentially with the number of inputs to each rule, we decided to use genetic programming to search for the best neurofuzzy structure to use.

3.4 Training

After the best neurofuzzy structure is determined, we train the network using the backpropagation algorithm, since we have differentiable T and S norms. Training the nuerofuzzy system will involve only adjusting the weights for the actual inputs to each neuron. This way we won't have situations in which we have multiple fuzzy variables are inputs to the same rule.

3.5 Overfitting Avoiding Technique

In training the neurofuzzy system, the training error and the testing error versus the training time decreases until we reach a stable minimal training error, however after a specific training time, the testing error will start to increase, which indicates that we are fitting the model to the training data, so we find the point after which the testing error starts to increase, and this will be training time that will give the best generalization error. Reducing the maximum number of hidden layer nodes also reduces overfitting. Neurofuzzy systems in general are less susceptible to over-fitting than neural networks. However the potential for over-fitting is there, and so we must find the best training time.

Chapter 4

Measurement data

4.1 Measurement process

A specialized Board Monitoring System (BMS) is installed on a bulldozer. A unit for data collecting, processing and storing is a central component of the BMS. The unit receives measurements from all sensors, performs some simple pre-processing before saving them. The location of the most important sensors together with all forces of interest is presented in Figure 19.

The sensors of the BMS are installed at different locations. The collected data are multi-dimensional. The measured attributes are:

1) Estimated amount of bulldozer load
2) Engine speed
3) Forward gear indicating a forward movement
4) Reverse gear indicating a reverse movement
5) Torque converter output
6) Track speed
7) Horizontal tension on the left and right sides of the dozer
8) Track pressures on the left and right sides of the dozer
9) Track tension on the left and right sides of the dozer
10) Torque
11) Tractive forces

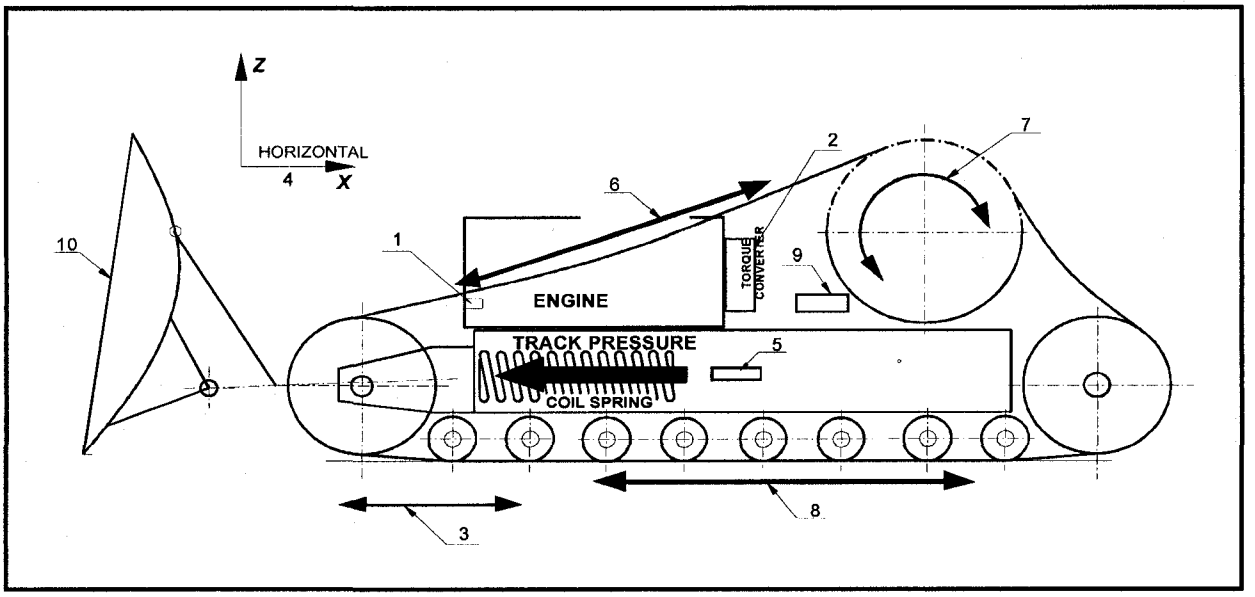


Figure 19: Dozer's schema with sensors and forces indicated

The bulldozer sensors are described below:

Engine speed sensor

Torque converter speed sensor

Track speed forward v>0 and reverse v<0

Horizontal tension of track (see item 6 below)

Pressure sensor (internal pressure from coil spring)

Track tension generated from applied torque

Torque (from engine, torque converter, transmission, differential and final drive)

Tractive forces, known as draw bar pool (DBP)

On-Board Data Collector system central unit of Board Monitoring System

Blade.

4.2 Data description

The measurement process took place during winter, and spring operating conditions. The bulldozer was performing levelling of overburden on a dumpsite.

4.2.1 *January dataset*

The data for January dataset contains 2485 data points. Table 2 shows data statistics, Table 3 shows the frequency distribution of the volume.

Table 2: January dataset statistics

Mean	67.90
Standard deviation	25.54
Skewness	-0.65

Table 3: Volume distribution for January dataset

Bin	Frequency	Cumulative %
10	78	3.14%
20	73	6.08%
30	92	9.78%
40	149	15.77%
50	233	25.15%
60	263	35.73%
70	339	49.38%
80	246	59.28%
90	409	75.73%
100	603	100.00%

4.2.2 *March dataset*

The March dataset data contains 1079 data points. Table 4 shows data statistics, table 5 shows the frequency distribution of the volume.

Table 4: March dataset statistics

Mean	78.36
Standard deviation	21.71
Skewness	-1.17

Table 5: Volume distribution for March dataset

Bin	Frequency	Cumulative %
10	7	0.65%
20	11	1.67%
30	32	4.63%
40	33	7.69%
50	71	14.27%
60	62	20.02%
70	86	27.99%
80	146	41.52%
90	201	60.15%
100	430	100.00%

4.3 Data cleaning

During pre-processing some of the attributes have been removed and modified. Two attributes: Forward Gear and Reverse Gear have been removed. Other six attributes representing pressures and tensions on both sides of the dozer have been merged. The merge process has been performed in pairs: left track tension and right track tension, left track pressure and right track pressure, as well as left horizontal tension and right horizontal tension have been averaged. As the result, nine attributes are used in the analysis. The names of these attributes with their brief description are presented in table 6.

Table 6: Attributes (features) of analyzed data and their brief description

Parameter	Unit	Description
Blade Volume	% of the bulldozer's blade	estimated amount of load pushed in front of the bulldozer in per cent of the fully loaded bulldozer's blade
Engine Speed	Rpm	revolution of engine
TC Out Speed	Rpm	torque converter output
Track Speed	km/h	relative velocity of track
Horizontal Tension	kPa	horizontal track tension – component of track tension
Track Pressure	kPa	internal pressure create tensions
Track Tension	kPa	track tension create from applied torque and resistance of the dozer
Torque	Nm	torque generate from power train
Forces	kN	tractive forces

Besides merging some attributes, and removing others, pre-processing of the data consisted of the removal of all data points with the value of blade volume equal to zero.

Chapter 5

Evolutionary based methods for system construction

Genetic algorithms and genetic programming can be used to search for the best possible structure of the neural or neurofuzzy network, like the number of hidden neurons for a neural network, or number of inputs, rules, and connections between the inputs and rules, and rules and outputs. Genetic algorithms can be used also for training weights of the neural network or the neurofuzzy system. Determing the structure of the system means knowing the rules that govern the behaviour of the system, and understanding the relationships between them, and the output. The genetic programming method used in this research is the grow method, where the length of the tree varies up to a maximum depth of D which is equal to the total number of AND's n the neurofuzzy model.

5.1 Genetic programming for system construction

The structure of the hybrid neurofuzzy system is determined using genetic programming, this includes:

1) Number of rules (AND neurons)
2) Number of inputs to each AND gates
3) The fuzzy inputs to each rule
4) The connection between the rules and the outputs

The function set is the T and S norms, and the terminal set consists of the fuzzy terms, and rule outputs.

The individuals in the population are created with varying number of AND's, so as to give the genetic programming system the ability to explore the best number of rules that can describe the system.

The number of inputs to each rule are generated randomly, since we have 8 variables in our research, and we require only 1 fuzzy term from each variable to be included in a rule, this is required so that we avoid situations in which we have a rule as

"If torque is high and torque is low then volume is high"

Since the system will pick some of the fuzzy terms available, the ones most important to the problem at hand, genetic programming is considered to be performing feature selection on the attributes, since it picks the most relevant ones, and discard the non important ones.

The connection between the rules and outputs is varied as well, as an example, if we have 3 output membership functions, then the first rule might be randomly chosen connected to output membership 1, or 2 and 3. The fact that the rule antecedent is connected to two different consequents does not mean that we have an inconsistency in our rules, this is because the connections have associated weights, so that the antecedent might have a weight of 0.6 towards output membership 2, but 0.1 towards output membership 3, so that the rule is contributing to different degrees towards each of the consequents. After the neurofuzzy structure is determined, the connection weights are trained using the backpropagation algorithm in chapter 2. The results are a set of rules that characterize the behaviour of the system using the most relevant variables.

5.1.1 Chromosome representation

Each chromosome in our population is represented by a linked list, in which each rule is represented by a structure that contains the outputs, the connections, and a pointer to another structure that contains the fuzzy inputs, and a pointer to the next rule, Figure 20.

```
struct node
{
 int    flag;
 char   string[MAX_SINGLE_STRING];
 int    coeff;
 node   *next;
};

struct main_node
{
 int        flag;
 int        child_counter;
 int        total_counter;
 double     fitness;
 double     worst;
 double     worst_tot;
 int        hits;
 int        no_of_outputs;
 int        outputs[OUTPUT_NO_MF];
 node       *child;
 main_node  *next;
};

struct main_node     *population [POPULATION_SIZE];
struct main_node *new_population [POPULATION_SIZE];
void (*pfitness)(int child);
```

Figure 20: Structure of neurofuzzy network

The main_node structure stores each of the rules of the system. The child counter contains the number of inputs to the rule. The total counter contains the total number of rules and their inputs. The no_of_outputs is the number of outputs the rule is connected to. The child is the pointer to the node structure which contains the fuzzy inputs. The fitness contains the fitness value for the neurofuzzy system, and the pointer next points to the next rule.

The node structure is where the children or fuzzy inputs to the neurofuzzy system are stored. The string array is where the actual fuzzy input is stored, string[0] contains the variable, and string[1] contains the membership function of the variable. If the coeff variable is set to 1, then we take the complement of the fuzzy term. The next pointer points to the next input to the rule.

5.1.2 Selection mechanism

Roulette wheel selection is the method by which we select individuals to undergo mutation or crossover. In this selection method, we generate a fitness value, and we compare it to the cumulative fitness of each individual, if the fitness value is lower than the cumulative fitness, then we select the individual. The individuals with a higher fitness value have a higher probability of being selected.

This method can represented using a wheel, we divide the wheel into slices, where the size of each slice corresponds to fitness of each individual, individuals who have a higher fitness get a large slice, and so the probability once we spin the wheel on landing on that slice is high.

5.1.3 Structure evaluation

The chromosomes in our population are evaluated using the stack. There are two stacks available, one for values, and one for operators. The chromosome value is calculated by first pushing and '+' operator which corresponds to the S-norm onto the operator stack, (number of inputs to that output membership - 1) times. So if we have 4 inputs to y1, then the '+' operator is pushed 3 times on the operator stack.

The program then iterates through the rules, and for each input to the rule, the program pushes the '*' operator, which represents the T-norm onto the operator stack (number of inputs to that rule – 1) times, so that if we have 3 inputs to R1, then the '*' operator gets pushed 2 times. The program then iterates over all inputs to the rule, and pushes their values onto the value stack. When there are 2 consecutive values in the stack, the program then, evaluates their T-norm, and pushes the result onto the stack again. The stack constantly shrinks as each rule is evaluated, and then outputs of all the rules are combined using the S-norms to calculate the value of the output membership. The program then iterates over the other outputs, repeating the process, Figure 21.

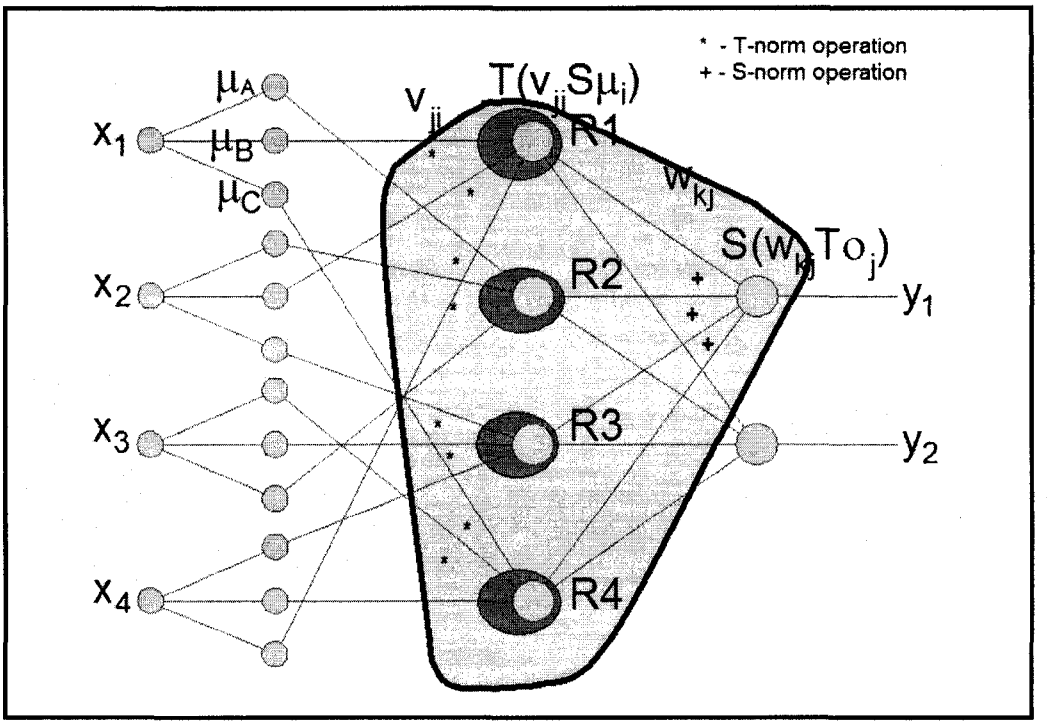


Figure 21: Neurofuzzy structure evaluation in genetic programming

When the chromosome is evaluated, the stack then will be at location 0.

5.1.4 Genetic programming operators

5.1.4.1 Crossover

Crossover is performed by selecting parent 1 and parent 2 using roulette wheel selection, and selecting the points of crossover for both parents. The points of crossover can either be a rule, which is indicated by selecting a main node, or a fuzzy term in one of the rules. The crossover point is determined randomly based on:

if (random value generated $< \frac{\text{maximum number of rules}}{\text{maximum number of rules} + \text{maximum number of inputs to each rule}}$),

then select a random rule as the crossover point, else select random input to random rule as crossover point

In the above rule for selecting the crossover point, if the number of rules allowed exceeds the number of inputs, the probability of mutating a rule is high, and the probability of selecting and inputs is low. In order to give equal probability to select and input, the same

as a rule, we need the number of allowed rules to be equal or close to the number of inputs.

Once the crossover points for both parents are determined, the crossover process is performed by combining the tree for parent1 from the root to the crossover point with the branch from parent 2 crossover point to the end of the tree as in Figure 22. The crossover process produces only one child.

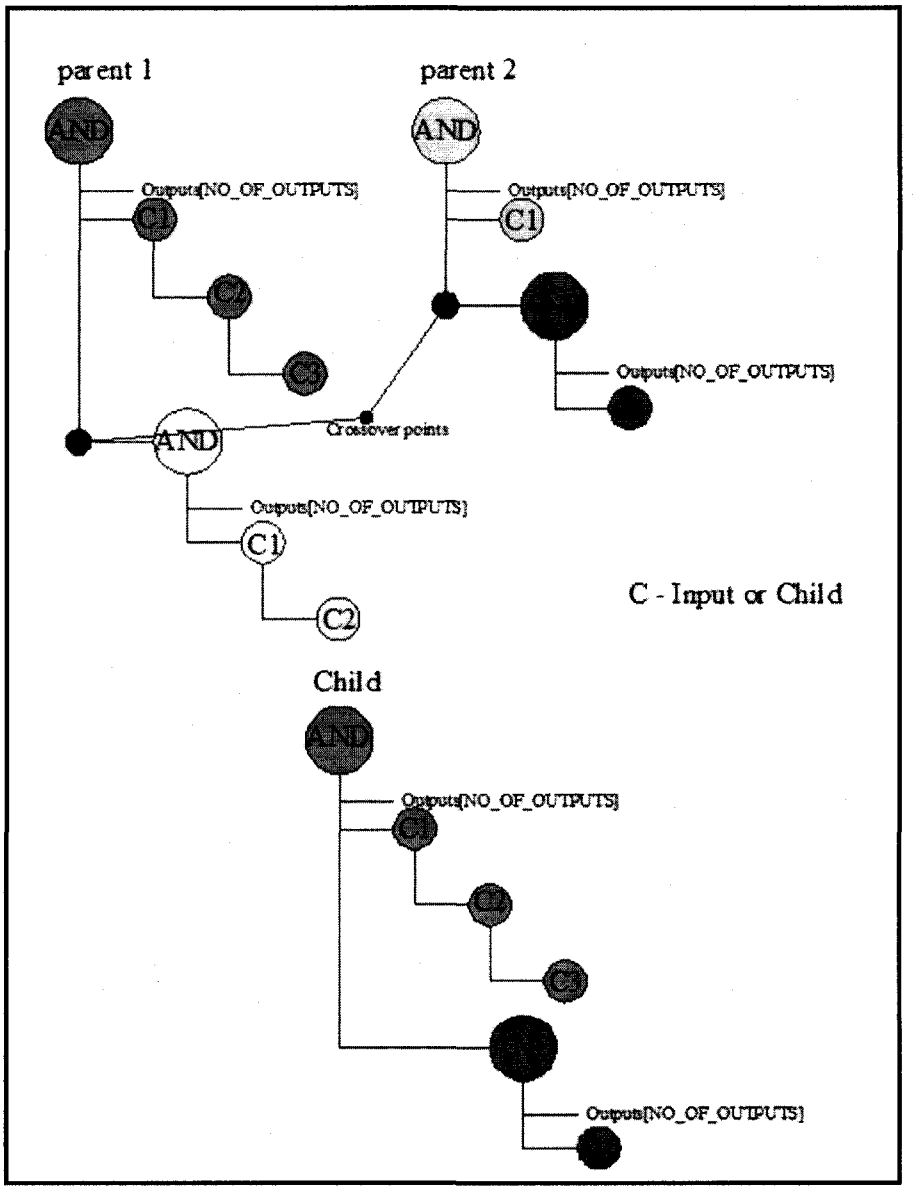


Figure 22: Chromosome crossover

The crossover function has to check to make sure that the maximum number of rules has not been exceeded, and if it has, the crossover process is repeated, selecting different crossover points, until we have a crossover, where the maximum number of rules has not been exceeded.

If the crossover point is a fuzzy term, then crossover is performed in the same way as for rules, except that we have to check to make sure that the maximum number of inputs to each rule has not been exceeded, and we have to make sure that each term appears only once in the rule inputs, regardless of its membership function, if any of these conditions are not satisfied, then we repeat the process selecting different crossover points. After the crossover is performed, then we check outputs from each rule, and if one of the outputs is not connected to any rules, we select a random rule, and increment its number of outputs by one, and we create a connection to that output.

5.1.4.2 Mutation

Mutation is done by first selecting a chromosome to mutate based on the mutation probability, and then selecting a mutation point, which could be either a rule, or an input to one of the rules in the structure.

The selection mechanism for a mutation point is the same as that for crossover, however, with one additional condition. If the child counter is equal to zero, then we select the main node for mutation. This allows us to initially generate the population using the mutation function, since the child counter is zero for newly initialized nodes.

If a rule has been selected for mutation, a new tree is generated and replaced with the sub tree at the mutation point. The mutation function checks if the maximum number of rules have been exceeded, and if it had, then mutation is repeated by selecting a different mutation point, and generating a new tree.

If the mutation point is an input for a rule, then, new inputs are generated, in which no variable is repeated twice, and replaced with the child tree of the mutation point, Figure 23.

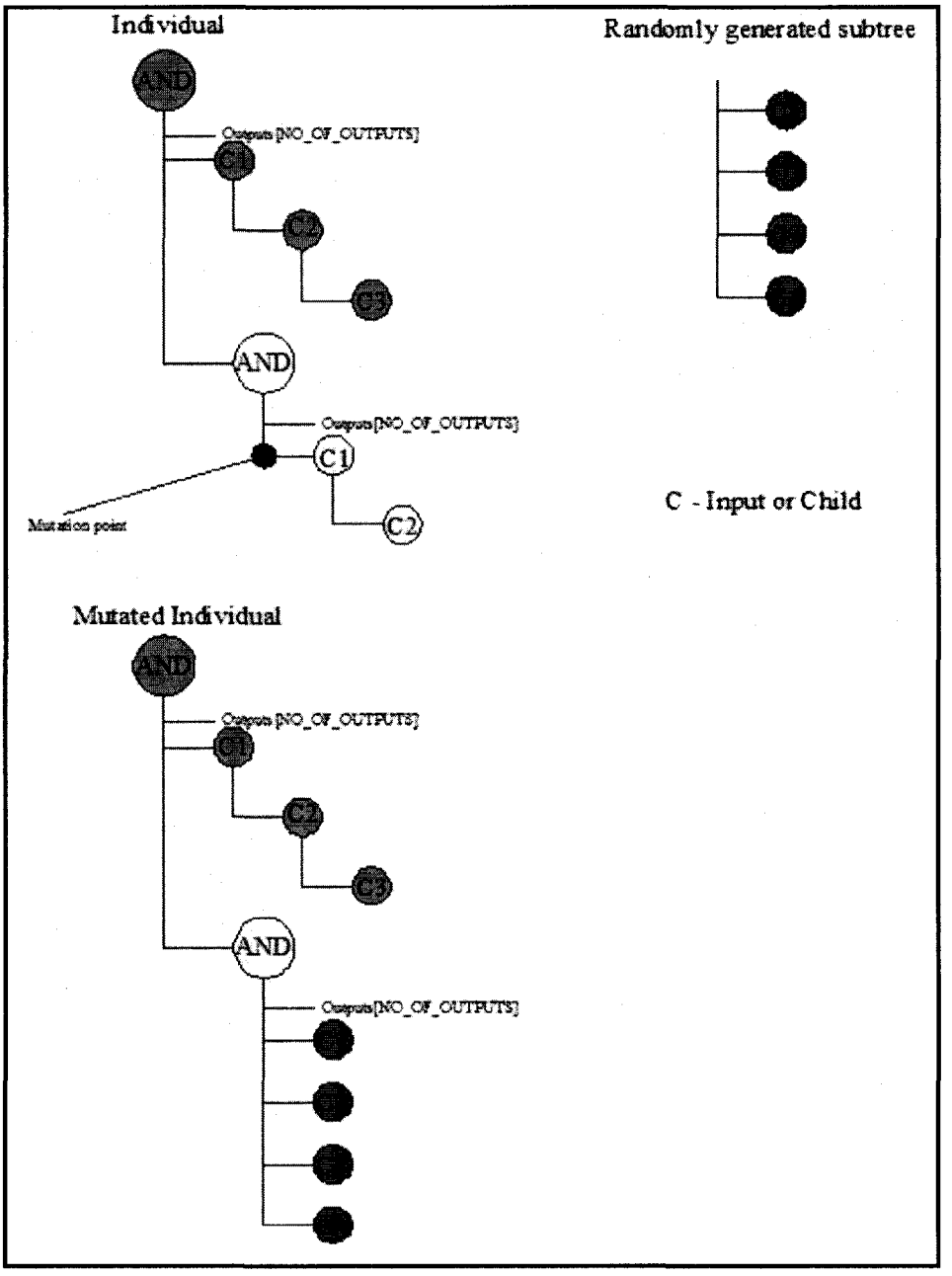


Figure 23: Chromosome mutation

5.1.5 Genetic programming parameters

Genetic programming parameters play a very important role in how much of the solution is space is explored, and how fast the genetic programming algorithm finds a solution. There are no specific guidelines as to the correct values of these parameters, and therefore they must be found my trail or error, or set based on the problem.

5.1.5.1 Population size

The population size controls the number of individuals in our population. It is kept constant through the evolution process. There is no guideline as to what the population size should be, however, if the population size is small; it will take longer for genetic programming to find a good solution, especially if the search space is large. For problems were we have a large search space, the population size should be large, in the hundreds or thousands.

5.1.5.2 Number of generations

The number of generations determines the evolution time of genetic programming, the longer the evolution time, the more of the solution space will be searched, and the more the chance we will find a good optimal solution. This value for large search spaces should be in the thousands of generations.

5.1.6 Fitness functions

The fitness functions in genetic algorithm, and genetic programming dictates the solution space to be search, and optimized. In genetic algorithms and genetic programming, we can change the fitness function easily to be more representative of the data, for example, we can use a fitness function that take the statistical distribution of the data into account. In our project, we have used the functions described in this section.

5.1.6.1 Sum squared error fitness function

The first type of membership that we have tried is the sum squared error function, defined by

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n} (y_{ij} - x_{ij})^2 \text{ , where}$$

k is the number of membership functions

n is the number of data points

y_{ij} is the desired membership output

x_{ij} is the actual membership output

This fitness function however, does not consider the distribution of the data into account. This fitness function is suitable for normal, or near normally distributed data.

5.1.6.2 Max Error function

Another fitness function is the maximum error over all output memberships.

$total\ error = \max(e_0, e_1, ..., e_n)$

where e_i is the maximum error for membership i over all training points

n is the number of output membership functions

This fitness function attempts to balance out the error across all membership functions.

5.1.6.3 Max error combination based function

In the maximum error combination fitness function, we determine the maximum error for each output membership function, and combined them to yield the error of the model. We combine the maximum errors using their product.

$$total\ error = \prod_{i=0}^{n} \max e_i,$$

where e_i is the maximum error for membership i over all training points

n is the number of output membership functions

In combining the errors from all membership functions, we try to minimize the error across all outputs.

5.1.6.4 Interval based error function

In the interval based error function, we divide the area under the membership function into intervals, and calculate the error and number of data points that lie in each interval. The intervals are shown in Table 7.

Table 7: Intervals and their corresponding ranges

Interval	Range of actual membership value
A	>= 0.75
B	>= 0.5 and < 0.75
C	>= 0.25 and < 0.5
D	>= 0.05 and < 0.25
E	>= 0 and < 0.05

The number of intervals we used for each membership function is 5. This will allow us to concentrate on the intervals that contain low number of data points. The error in each interval is normalized by the number of data points in there interval, the total error is calculated by:

$$total\ error = \sum_{i=A}^{E} (1 + \Delta w_i) * \frac{error_i}{n_i}$$, Where n is the number of points in interval i

Δw_i is initialized to 0 for all intervals, and then updated every 20 generations. Δw_i is updated by sorting the error for all intervals from largest to smallest, the largest error having a rank of 0 and the lowest having a rank of 4, and then updating the weights for each interval in proportion to the error ranking

$$\Delta w_i = \frac{(5 - Rank)}{5} * error_i$$

Chapter 6

Results and Discussions

The system was run for both the January dataset and March dataset. In order to determine the best fitness function to use for our final model, for both the January dataset and March dataset, we have used 10 fold cross validation for improved accuracy results. The results have been averaged for each fitness function for both the January and March datasets. The systems were run with the following parameters:

1) Generation size is 1000
2) Population size is 100
3) Probability of crossover is 0.8
4) Probability of mutation is 0.2
5) Number of input and output membership functions is 3

The number of memberships was chosen to be 3 memberships because it limits the search space, and because it produces the lowest absolute average error.

The reason for using high crossover and mutation rates is because of the low number of population, and so to create as much variation in the population as possible, we increase both mutation and crossover rates.

For each of January dataset and March dataset, the fitness functions were compared, and the one with the lowest absolute average error is selected as the most appropriate fitness function for the data, the data is then run 3 fold cross validation with 66% of the data for training, and 33% for testing, and the following parameters:

1) Generation size is 2000
2) Population size is 500
3) Probability of crossover is 0.4
4) Probability of mutation is 0.05

6.1 Results

6.1.1 January dataset

The data is fuzzified using the equations in section 3.3, and fed into the system. The results from 10 fold cross validation for January dataset is shown in table 8.

Table 8: January dataset cross validation results (Before defuzzification)

Fitness	Sum squared error	Mean sum squared error
SSE	1.8450	0.0074
Max error function-product	4.6672	0.0187
Max error function	21.8643	0.0878
Interval based error	2.6217	0.0105

From table 8, we can see that the lowest error is achieved with the sum squared error function, second to that is the interval based error, and lastly the max error function.

The best system from the 10 fold cross validation results for the SSE fitness function for January dataset data contains the following 12 fuzzy terms, table 9.

Table 9: Fuzzy terms used by the best program in the 10 fold cross validation run

Fuzzy Variable	Explanation
E2	Engine speed is medium
E3	Engine speed is high
S1	Track speed is low
S3	Track speed is high
H1	Horizontal track tension is low
H2	Horizontal track tension is medium
T2	Track tension is medium
Q1	Torque is low
Q2	Torque is medium
Q3	Torque is high

F1	Forces are low
F2	Forces are medium
F3	Forces are high

6.1.1.1 Prediction System for "January dataset" Data

The system was run again with 3 fold cross validation, the best program out of the 3 fold cross validation produced the results in table 10.

Table 10: Defuzzified results of best program in the 3 fold cross validation run

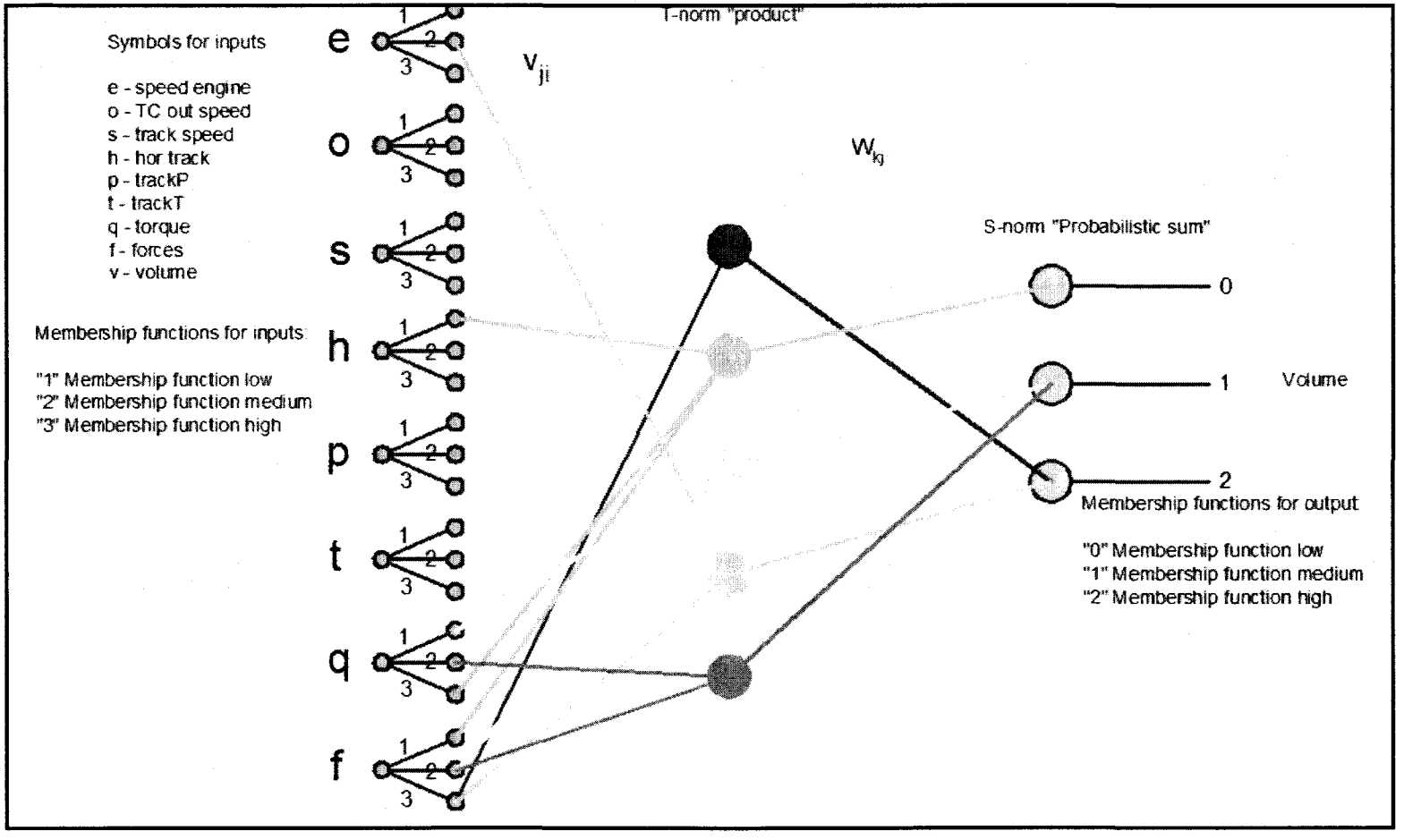
Absolute average error	Standard deviation
2.9595	2.0510

The January system was run using the test data from the March dataset, and the results are summarized in table 11.

Table 11: Test sets results (without defuzzification)

	Training	Testing
January	0.00516	0.00533
March	-----	0.01145

Figure 24 shows the structure of the neurofuzzy system.


Symbols for inputs
e - speed engine
o - TC out speed
s - track speed
h - hor track
p - trackP
t - trackT
q - torque
f - forces
v - volume
Membership functions for inputs
"1" Membership function low
"2" Membership function medium
"3" Membership function high
e
o
s
h
p
t
q
f
1
2
3
T-norm "product"
v_{ji}
w_{kj}
S-norm "Probabilistic sum"
0
1
2
Volume
Membership functions for output
"0" Membership function low
"1" Membership function medium
"2" Membership function high

6.1.1.2 Knowledge gained from January dataset system

The structure of the neuro-fuzzy systems allows for extraction of IF-THEN rules that represent the relations between input parameters (Table 11) and the volume of bulldozer's overburden. The rules are presented below. Please note that each rule of the form

IF
 {(VARIABLE$_1$ is *low/medium/high*)$_0$ and (VARIABLE$_2$ is *low/medium/high*)$_{0.6}$ }$_{0.9}$
OR
 {(VARIABLE$_4$ is *low/medium/high*)$_{0.3}$ }$_{0.6}$
THEN
 VOLUME is low/medium/high

Should be read in the following way:

> The contribution of a single antecedent (between {} brackets) to the statement "VOLUME is *low/medium/high*" is 0.9, and the contributions of the statements (VARIABLE$_1$ is *low/medium/high*) and (VARIABLE$_2$ is *low/medium/high*) to this antecedent are 0.0 and 0.6 respectively[1].
> while the contribution of the second antecedent to the statement "VOLUME is *low/medium/high*" is 0.6, while the contribution to this antecedent coming from the (VARIABLE$_4$ is *low/medium/high*) is 0.3.

The system leads to the set of the following rules:

VOLUME is low:

IF
{
(HORIZONTAL TRACK TENSION is low)$_{0.95}$ and (TORQUE is low)$_{0.94}$ and (FORCES are low)$_0$
}$_1$

VOLUME is medium:

IF
{
HORIZONTAL TRACK TENSION is medium)$_0$ and (TORQUE is medium)$_0$ and (FORCES are low)$_0$
}$_{0.29}$

---

[1] In the "and" part of the fuzzy IF-THEN rule, the smaller value of a weight means higher contribution, while the higher value of a weight means lower contribution – this is due to t-norm.

OR

IF
{
(FORCES are medium)$_0$ and (TORQUE is low)$_{0.33}$
}$_1$

OR

IF
{
(FORCES are medium)$_0$ and (TORQUE is medium)$_0$
}$_{0.96}$

VOLUME is high:

IF
{
(FORCES are high)$_0$
}$_1$

OR

IF
{
(ENGINE SPEED is medium)$_{0.47}$ and (FORCES are high)$_0$
}$_{0.79}$

Applying a threshold of 0.5, so that if the weight of any input to a rule is lower than 0.5, then we completely include the input, or if the input to the OR is bigger than 0.5 we can include the input to the OR, otherwise we discard, the rules reduce to:

VOLUME is low:

IF
{
(FORCES are low)$_0$
}$_1$

VOLUME is medium:

IF
{
(FORCES are medium)$_0$ and (TORQUE is low)$_0$
}$_1$

OR

IF
{
(FORCES are medium)$_0$ and (TORQUE is medium)$_0$
}$_1$

VOLUME is high:

IF
{
(FORCES are high)$_0$
}$_1$

OR

IF
{
(ENGINE SPEED is medium)$_0$ and (FORCES are high)$_0$
}$_1$

The rimpull force is a very important factor, since it is always fully included in the rules, and the fact that it appears alone and directly related to the volume.

6.1.2 March dataset

The results from 10 fold cross validation for March dataset is shown in table 12.

Table 12: March dataset cross validation results (Before defuzzification)

Fitness	Sum squared error	Mean sum squared error
SSE	1.1514	0.0107
Max error function -product	7.3587	0.0681
Max error function	7.0308	0.0651
Interval based error	0.8611	0.0079

From Table 12 we can see that the lowest error is achieved with the interval based error function, second to that is the sum squared error function, and lastly the max error function –product.

The best system from the 10 fold cross validation results for the interval based fitness function for March dataset data contains the following 11 fuzzy terms, table 13.

Table 13: Fuzzy terms used by the best program in the 10 fold cross validation run

Fuzzy Variable	Explanation
S1	Track speed is low
S2	Track speed is medium
S3	Track speed is high
H1	Horizontal track tension is low
T3	Track tension high
Q1	Torque is low
F1	Forces are low
F2	Forces are medium
F3	Forces are high

6.1.2.1 Prediction System for "March dataset" Data

The system was run again with 3 fold cross validation, the best program the best program out of the 3 fold cross validation produced the results in table 14.

Table 14: Defuzzified results of best program in the 3 fold cross validation run

Absolute average error	Standard deviation
2.2955	1.7760

The prediction system was run using the test set of the January dataset, and the results are summarized in table 15.

Table 15: Test sets results (without defuzzification)

	Training	Testing
March	0.00822	0.00803
January	-------	0.00798

Figure 25 shows the structure of the neurofuzzy system.

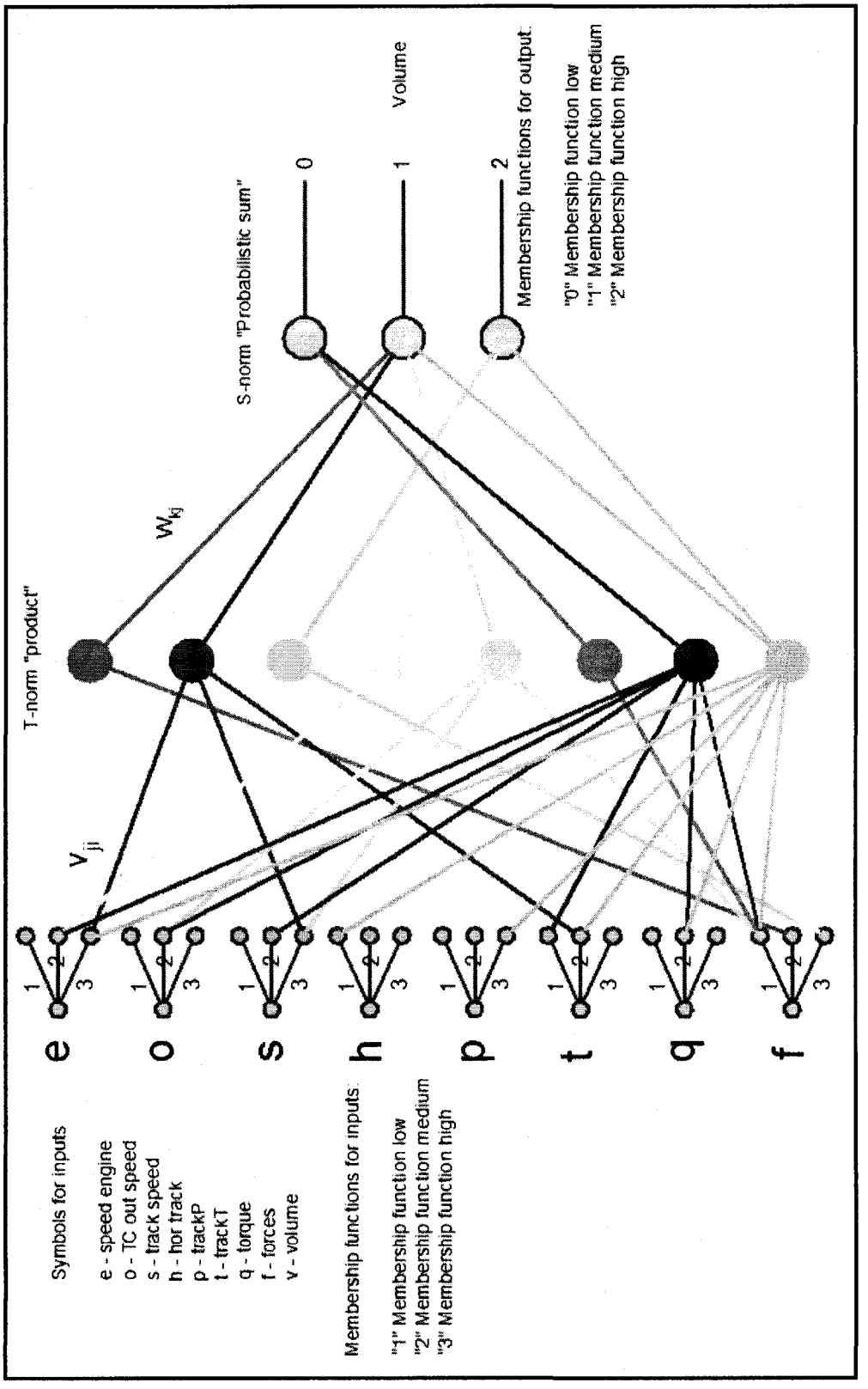


Figure 25: The structure of March dataset

6.1.2.2 Knowledge gained from March dataset system

The following rules have been extracted from the system constructed for the March dataset:

VOLUME is low:

IF
{
(FORCES are low)$_0$
}$_1$

OR

IF
{
(TRACK SPEED is low)$_0$ *and (TORQUE is medium)*$_0$ *and (TRACK TENSION is low)*$_0$ *and (ENGINE SPEED is medium)*$_0$ *and (TORQUE CONVERTER SPEED is medium)*$_0$*and (FORCES are low)*$_0$
}$_1$

VOLUME is medium:

IF
{
(FORCES are medium)$_0$
}$_{0.8837}$

OR

IF
{
(ENGINE SPEED is high)$_0$ *and (TRACK TENSION is medium)*$_0$ *and (TRACK SPEED is high)*$_0$
}$_1$

OR

IF
{
(TRACK SPEED is high)$_0$ *and (TORQUE CONVERTER SPEED is medium)*$_0$ *and (FORCES are low)*$_0$
}$_1$

OR

IF
{
(ENGINE SPEED is high)$_0$ and (TRACK TENSION is medium)$_0$ and (TRACK PRESSURE is high)$_0$ and (TORQUE is medium)$_0$ and (FORCES are low)$_0$ and (HORIZONTAL TRACK TENSION is low)$_0$
}

VOLUME is high:

IF
{
(FORCES are high)$_0$
}$_1$

OR

IF
{
(FORCES are high)$_0$ and (ENGINE SPEED is medium)$_{0.292}$
}$_{0.837}$

OR

IF
{
(ENGINE SPEED is high)$_0$ and (TRACK TENSION is medium)$_0$ and (TRACK PRESSURE is high)$_0$ and (TORQUE is medium)$_0$ and (FORCES are low)$_0$ and (HORIZONTAL TRACK TENSION is low)$_0$
}

After applying the threshold of 0.5:

VOLUME is low:

IF
{
(FORCES are low)$_0$
}$_1$

OR

IF
{
(TRACK SPEED is low)$_0$ and (TORQUE is medium)$_0$ and (TRACK TENSION is low)$_0$ and (ENGINE SPEED is medium)$_0$ and (TORQUE CONVERTER SPEED is medium)$_0$and (FORCES are low)$_0$
}$_1$

VOLUME is medium:

IF
{
(FORCES are medium)$_0$
}$_{0.8837}$

OR

IF
{
(ENGINE SPEED is high)$_0$ and (TRACK TENSION is medium)$_0$ and (TRACK SPEED is high)$_0$
}$_1$

OR

IF
{
(TRACK SPEED is high)$_0$ and (TORQUE CONVERTER SPEED is medium)$_0$ and (FORCES are low)$_0$
}$_1$
OR

IF
{
(ENGINE SPEED is high)$_0$ and (TRACK TENSION is medium)$_0$ and (TRACK PRESSURE is high)$_0$ and (TORQUE is medium)$_0$ and (FORCES are low)$_0$ and (HORIZONTAL TRACK TENSION is low)$_0$
}

VOLUME is high:

IF
{
(FORCES are high)$_0$
}$_1$

OR

IF
{
(FORCES are high)$_0$ and (ENGINE SPEED is medium)$_0$
}$_1$

OR

IF
{
(ENGINE SPEED is high)$_0$ and (TRACK TENSION is medium)$_0$ and (TRACK PRESSURE is high)$_0$ and (TORQUE is medium)$_0$ and (FORCES are low)$_0$ and (HORIZONTAL TRACK TENSION is low)$_0$
}

The system did not exclude any rules or variables, however, the rimpull force is still an important factor, since it appears in almost all rule inputs with a high weight, and appears alone with direct relation to the volume.

6.1.3 Comparison of Results

The results from both January dataset and March dataset show that the rimpull force is an important factor that influences the volume of the bulldozer, some other factors which are common to both January dataset and March dataset like:

1) Engine speed
2) Torque

The results from using the March test set in the January system, and the January test set in the March system indicate that the March system can also be used for the January data, but not the opposite.

Chapter 7

Conclusion and Recommendations

The research conducted in the framework of this thesis is quite innovative. The issue of predicting the volume of material pushed in the front of a bulldozer has not been addressed so far. The procedures for processing and modeling of data proposed and validated in the thesis can be treated as a prototype of methodology that can be applied to processing and modeling of any data collected during operations of heavy earth-moving vehicles.

In particular, the contributions of the thesis are as follows:

collection of on-site measurements of heavy earth-moving equipment, performed during January and March;

processing – cleaning and initial analysis – of the measurement data sets; this task has been preformed in order to ensure consistency of data points; validation of data has been done;

development of neuro-fuzzy models of the measurement data using an evolutionary-based optimization system.

Overall, the research activities have resulted in construction of systems for predicting the volume of material pushed by the bulldozer. Analysis of two best models developed for both – January data and March datasets has lead to the conclusion that the rimpull force is the major factor in predicting the volume of material. Most of measured quantities constitute inputs to both systems. However, the rules extracted from the systems are

different. This means that relationships between measured quantities and the volume are different for different measurement conditions (times when measurements were done).

The presented results are a part of studies dedicated to construction of a system capable of predicting force distributions during on-site operations of heavy earth moving machinery. To predict distribution of forces – in particular forces reacting on wheels – the volume of material pushed in the front of bulldozer has to be know.
In order to gain confidence in the proposed approaches suitable for data analysis and modeling an additional set of experiments should be conducted. They should target more variety of measurements – different weather conditions and different tasks performed by vehicles, and more detailed analysis of extracted rules. All this should lead to construction of a more comprehensive force distribution prediction system. More experiments with modifications of the proposed evolutionary-based optimization process – more tuning towards nature of the measured data – should be conducted.

References

[1] Brown, M., An Introduction to Fuzzy and Neurofuzzy Systems, 1996.

[2] Hirota, K., Pedrycz, W., OR/AND Neuron in Modeling Fuzzy Set Connective, *IEEE Transactions on Fuzzy Systems,* Vol. 2, No. 2, May 1994, pages: 151-161.

[3] Hirota, K., Pedrycz, W., Fuzzy Computing for Data Mining, *Proceedings of the IEEE*, Vol. 87, No. 9, Sep 1999, pages: 1575-1600.

[4] Iskarous, M., Kawamura, K., Intelligent Control Using a Neuro-Fuzzy Network, *Proceedings of the International Conference on Intelligent Robots and Systems*, Vol. 3, 1995, pages 3350-

[5] ISO/SAE 6155 – *Earth-moving machinery – Basic types – Vocabulary.*

[6] ISO/SAE 9246 – *Earth-moving machinery – Crawler and wheel tractor dozer blades – Volumetric ratings.*

[7] Koza, J. Z., *Genetic Programming as a means for programming computers by natural selection*, Springer, 1994.

[8] Kumar, R., and Kumar, U., Service Delivery Strategy: Trends in Mining Industries, *International Journal on Surface Mining, Reclamation and Environment*, Vol. 18, 2004, pages 299-307.

[9] Nadjeh, N., de Macedo Maurelle, L., *Fuzzy Systems Engineering*, Springer-Verlag, New York, 2005.

[10] Paraszczk, J., Planeta, S., and Szymanski, J., Performance and Efficiency Measures for Mining Equipment, *Proc. of the 9th Inter. Symposium on Mine Planning and Equipment Selection*, Athens/Greece, November 6-9, 2000, pages 667-672.

[11] Pedrycz, W., Gomide, F., *Fuzzy Systems Engineering: Toward Human-Centric Computing*, Wiley-IEEE Press, 2007.

[12] Pedrycz, W., Reformat, M., Li, K., OR/AND Neurons and the Development of Interpretable Logic Models, *IEEE Transactions on Neural Networks*, Vol. 17, 2006, pages 636-658.

[13] Scoble, M.J., Peck, J., and Hendricks, C., A Study of Surface Mine Equipment Monitoring, *International Journal of Surface Mining and Reclamation*, 1991, Vol. 5, pages 111-116.

[14] Zelenin, A.N., Balovnev, V.I., and Kerov, I.P., *Machines for Moving the Earth*, Oxonian Press, New Delhi, 1985.

[15] http://en.wikipedia.org/wiki/Donald_Olding_Hebb

[16] http://en.wikipedia.org/wiki/Paradox_of_the_heap

[17] http://www.swarthmore.edu/NatSci/echeeve1/Ref/HH/index.htm

[18] http://en.wikipedia.org/wiki/Triangular_norm

[19] http://en.wikipedia.org/wiki/Triangular_norm