CHAPTER 5

CHANGE DETECTION IN MIGRATING PARALLEL WEB CRAWLER: A NEURAL NETWORK BASED APPROACH

5.1 Introduction

Web pages change constantly, a web crawler periodically download fresh pages as well as it refresh downloaded pages, so that the database is maintained fresh. Web crawler revisit downloaded pages to maximize the freshness of web page. Web pages tend to change after certain time interval. Different web pages changes at different time interval, the “freshness” is maintained by downloading the web pages by revisiting them. If Web documents changes at similar rate, then the web crawler visits the pages at the constant rate.

5.2 Change Detection Algorithms

Changes are gathered in a delta script or delta file. The delta script is used to apply the changes and recreate the changed file again. The changes are also called operations and the most common operation types are insert and delete. Most algorithms specify updates, which are combination of delete and insert. Less common is move which is also combination of one delete and one insert. This section provides an overview of different change detection algorithms. In [147], the authors proposed hierarchical structured information which is a structural differencing tool. It takes two versions of a LaTeX document as input and produces a LaTeX document as output file. In an ordered tree FastMatch algorithm is used which uses a function equal to compare nodes. To perform initial matching of nodes that appear in the same order longest common subsequence computations is used by FastMatch. In [148], the authors demonstrated the cost of the algorithm is $O(n^*e+e^3)$, where $n$ is the number of leaf nodes and $e$ is the sum of the number of deleted and inserted subtrees. This algorithm only supports certain
LaTeX elements; hence for change detection in XML documents FastMatch cannot be used. In [149], the authors proposed change detection in hierarchical structured data trees. It demonstrated the changes between two trees in a comprehensive manner. Script is used that gives the sequence of operations needed to transform the original tree into the new tree. The algorithm provides the basic functionality of updating, inserting and deleting along with copy and move operations. In [147], the authors transformed the problem to the edge cover problem. It provides a heuristic solution in worst-case time of $O(n^3)$ time and an average time of $O(n^2)$. In [151], the authors stated that time complexity of algorithm is of $O(n^2 \log n)$.

In [150], authors allowed both program access and command line, is a set of Java beans. The algorithm is designed to perform fast differentiation and update of Document Object Model structures. The advantage of DOM is that it is simple to use but it is slow and cannot handle documents of large sizes. Differences are expressed as update, delete and insert operations. The cost of algorithm is $O(n^2)$ time, where $n$ is the number of tree nodes. The algorithm uses an optimal tree-differentiating algorithm and a fast subtree matching procedure. In [152], authors proposed an algorithm based on main memory for calculating a minimum-cost edit script between two trees that are rooted and ordered. The algorithm constructs a matrix based on problem of string edit. In [153], authors proposed that the algorithm is a good choice when the requirements are minimum of the result, but when the computation cost is high.

In [154], author developed a Java program, which compares a base XML document and another XML document. The algorithm points out the differences by use of symbols and color. In [155], authors pointed out that the algorithm is unable to produce correct result, as developers made the incorrect assumption that if all children of the same node have no conflicts then the node is free of conflicts. In [156], authors described the differencing and 3DM merging tool for XML, it is a XML tree matching algorithm which is a change detection algorithm. The algorithm handles ordered tree. The algorithm depicts that “the issue of constructing matching’s efficiently, effectively and accurately is of great importance when implementing XML merging for documents without unique element identifiers”. Some operations that the algorithm supports are delete, insert, update, move and copy.
In [148], the authors proposed a tool for differencing ordered XML trees for improving time and memory management. In [157], the authors developed a prototype of a dynamic warehouse for XML data. The algorithm computes hash values and weights for every node in the XML trees of both XML documents. For each node in the original document there is a unique identifier. The identification technique used in the prototype gives the list of identifiers in the XML document. In [158], the authors proposed the algorithm that uses the format in which each XML document consists of all forward completed deltas, in order to save the detected changes. The algorithm compares the signatures of two nodes and if they are equal, the nodes are matched. The algorithm supports the insert, delete, update and move operations with time complexity of $O(n \log n)$.

5.2.1 Types of Change

There are following types of change:

- Non-logical change: e.g. in a natural language description. These changes are in the label of a concept or property, or in a comment inside a definition.

- Logical definition change: These changes are in the definition of a concept or property that affects its formal semantics. For examples such changes are alterations of domains.

- Identifier change: These changes are when a concept or property is given a new identifier, i.e. a renaming.

- Addition of definitions.

- Deletion of definitions.

Most of these changes can be detected completely, except for the identifier change, because this change is not distinguishable from a subsequent deletion and addition of a simple definition.
5.2.2 Detecting Changes

Main problem with the detection of change is the abstraction level at which changes should be detected. Abstraction is required to distinguish between changes in the representation that affect the meaning and those that don’t influence the meaning. Detecting changes in the representation is insufficient; change should also be detected in the content and structure of the web page. However considering the logical meaning only is not enough abstracting too far can also be a problem. Finding the right level of abstraction is important. Second problem with detection of change is even when the correct level of abstraction for change detection is found; the conceptual implication of such a change is not yet clear.

5.3 Background on Neural Networks

Artificial neural networks (ANNs) resemble the processing capabilities of the human brain. The components of neural network are computational units similar to the neurons of the brain. A neural network is formed from one or more layers of neurons. Each neuron in the network performs calculations that contribute to the overall learning process of the network. The neural network is parallel information processing system that learn and store knowledge about its environment. The two factors that influence the performance of the neural network are its parallel distributed design and its capability to extrapolate the learned information. Data mining, pattern recognition and function approximation are tasks that can be performed by neural networks. In our technique, the neural network is used to produce a output when an input values are given. The neurons of the network are composed of: a set of n synaptic weights, an adder and an activation function. If the input signal to neuron i is x_j , the synaptic weight associated with the interconnection between the input signal and the neuron is denoted w_{ij} . A multi-layer feed forward neural network consists of an input layer of non computational units (one for each input), there are one or more hidden layers of computational units, and an output layer of computational units. Backpropagation is the standard training method that is applied to multi-layer feedforward networks. The algorithm consists of passing the input signal forward and the error signal backward through the network. Multi-layer
feed forward neural networks are capable of solving complex problems, and the backpropagation technique is efficient as a training method.

5.4 Neural Network Based Model For Change Detection Of Web Pages

About 60% of Web content is dynamic in nature [118] and 14% of the links in search engines are broken [116]. It is necessary to refresh the database of web pages at regular interval of time. The focus should be on updating only those documents of the database which has actually changed. Pages are added, deleted, removed, rearranged and modified every other second on the Internet, these modifications take place frequently. This changing nature of the World Wide Web is one of the major concerns in designing the Migrating Parallel web crawler. The rate of change varies from site to site. Therefore, managing the local collection afresh becomes a challenging task. The changes that can occur in web pages are Structural Changes, Content level changes, Behavioral and Presentation changes. The mechanism for change detection use the policy of retaining copies of web pages cached which are compared with downloaded web document afterwards to determine if there has been any change or not. Here neural network is used to detect change in web pages. Proposed Algorithm is implemented in MATLAB using Neural Network Tool box.

5.4.1 Neuron Model

A linear neuron model with R inputs \((p_1, p_2, p_3, \ldots, p_R)\) is shown below.

\[
\alpha = \text{purelin}(W p + b)
\]

Where...

- \(R = \text{number of elements in input vector}\)
The basic structure of this network is same as the perceptron. Linear neuron uses purelin as linear transfer function shown in figure 5.2. The values passed are returned to the linear transfer function that calculates the output of neuron. Where \( a = \text{purelin}(n) = \text{purelin}(W_p + b) = W_p + b \). The neurons are trained to calculate a linear approximation to a nonlinear function or an affine function of its inputs. To perform a nonlinear computation a linear network cannot be used.

### 5.4.2 Network Architecture

The linear network has a layer of neurons denoted by \( S \) connected to \( R \) \((p_1, p_2, p_3, \ldots, p_R)\) inputs through a matrix of weights \( W \). The linear network is shown below.

![Network Architecture Diagram](image-url)
A linear network with single layer is shown above. The capability of this network is same as multilayer linear networks. Every multilayer linear network can have an equivalent single-layer linear network.

5.5 Neural Network Based Change Detection Of Web Pages

Change in structure
- Initially the graph is derived from web pages.
- Adjacency matrix A is obtained at any time t and Adjacency matrix B is obtained at any time t’ from graph.
- Using MATLAB for detecting change in the structure.
- Using Neural Network tool box for creating neural network
- Assign input vector (v) values
- Assign target values (t)
- Create neural network
- Training the neural network
- The result of training
- Simulating the neural network
- Analysis of result obtained.

Change in contents
- Change in Content vector V is obtained.
- Using MATLAB for detecting change in the content.
- Using Neural Network tool box for creating neural network
- Assign input vector (v) values
- Assign target values (t)
- Create neural network
- Training the neural network
- The result of training
- Simulating the neural network
- Analysis of result obtained.
5.6 Description

5.6.1 Method Followed For Detecting Change In The Structure

1. Initially the graph G1 and G2 are derived from web pages.

2. Adjacency matrix A is obtained from graph G1 and Adjacency matrix B is obtained from graph G2.

   \[
   A = \begin{bmatrix}
   0 & 0 & 1 & 1 & 0 \\
   1 & 0 & 0 & 0 & 0 \\
   0 & 0 & 0 & 0 & 1 \\
   0 & 0 & 0 & 0 & 0 \\
   0 & 1 & 1 & 0 & 0 \\
   \end{bmatrix}
   \]

   \[
   B = \begin{bmatrix}
   0 & 1 & 0 & 0 & 0 \\
   1 & 0 & 0 & 0 & 1 \\
   1 & 0 & 0 & 0 & 0 \\
   0 & 1 & 0 & 0 & 1 \\
   0 & 1 & 0 & 1 & 0 \\
   \end{bmatrix}
   \]

3. MATLAB is used for detecting change in the structure.

   a) The Neural Network tool box is used for creating neural network

   b) Assign input vector (v) values
Figure 5.4: Assign Input values to Neural Network

c) Assign target values (t)

Figure 5.5: Assign target values to Neural Network
d) Create neural network with following:

![Image of neural network creation process]

**Figure 5.6: Neural Network description**

i) Name: Changedetection

ii) Network properties

1) Network Type: Feed forward Back propagation
2) Input Range: [0,1;01]
3) Training Function: TRAINLM
4) Adaptation Learning Function: LEARNGDM
5) Performance function: MSE
6) No of Layers: 2

**trainlm**: trainlm is the Levenberg-Marquardt back propagation. According to Levenberg-Marquardt optimization the network training function that updates weight and bias values is trainlm. The syntax is trainlm(net, Pd,Tl,Ai,Q,TS,VV,TV) with following parameter: Neural network is denoted by net, Delayed input vectors are denoted by Pd, Layer target vectors are denoted by Tl, Initial input delay conditions are denoted by Ai, Batch size is denoted by Q, Time steps are denoted by TS, An empty matrix [] or a structure of validation vectors are denoted by VV, An empty matrix [] or a
structure of test vectors are denoted by TV and returns net which is Trained neural network. The training record is denoted by TR. The Epoch number is TR.epoch, Training performance is TR.perf, Validation performance is denoted by TR.vperf, The Test performance is given by TR.tperf, Adaptive mu value is denoted by TR.mu. Training of the neural network occurs according to training parameters of trainlm. Default values are:

<table>
<thead>
<tr>
<th>Training Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epochs</td>
<td>100</td>
<td>epochs to train</td>
</tr>
<tr>
<td>goal</td>
<td>0</td>
<td>Performance goal</td>
</tr>
<tr>
<td>max_fail</td>
<td>5</td>
<td>Maximum validation failures</td>
</tr>
<tr>
<td>mem_reduc</td>
<td>1</td>
<td>memory/speed tradeoff</td>
</tr>
<tr>
<td>min_grad</td>
<td>1e-10</td>
<td>Minimum gradient</td>
</tr>
<tr>
<td>mu</td>
<td>0.001</td>
<td>Initial mu</td>
</tr>
<tr>
<td>mu_dec</td>
<td>0.1</td>
<td>mu decrease factor</td>
</tr>
<tr>
<td>mu_inc</td>
<td>10</td>
<td>mu increase factor</td>
</tr>
<tr>
<td>mu_max</td>
<td>1e10</td>
<td>Maximum mu</td>
</tr>
<tr>
<td>show</td>
<td>25</td>
<td>Epochs between displays</td>
</tr>
</tbody>
</table>

Table 5.1: Default values of trainlm's training parameters

Dimensions for these variables are: Tl is Ni x TS cell array where each element Tl{i,ts} is a matrix of dimension Vi x Q, Pd is No x Ni x TS cell array where every element of Pd{i,j,ts} is a matrix of dimension Dij x Q, Ai is Ni x LD cell array where each element Ai{i,k} is a matrix of dimension Si x Q. Where Ni denotes net.numLayers, Ni denotes net.numInputs, LD denotes net.numLayerDelays, Si denotes net.layers{i}.size, Ri denotes net.inputs{i}.size, Dij denotes Ri * length(net.inputWeights{i,j}.delays), Vi denotes net.targets{i}.size, If VV or TV is not [], it must be a structure of vectors, where VV.Tl, TV.Tl are validation/test layer targets, VV.PD, TV.PD are validation/test delayed inputs, VV.Ai, TV.Ai are validation/test initial input conditions, VV.TS, TV.TS
are validation/test time steps, VV.Q, TV.Q are validation/test batch size, trainlm(code) returns information for each code string where pdefaults denotes default training parameters, pnames denotes names of training parameters.

The trainlm can train network on the basis of its weight, transfer functions and net input. trainlm assumes that the performance function of network is mse. jX the Jacobian of perf performance with respect to the bias variables X and weight is calculated by Backpropagation. Levenberg-Marquardt adjust each variable according to following,

\[ jj = jX \cdot jX \] //where X is bias variable

\[ je = jX \cdot E \] // where E denotes all errors

\[ dX = -(jj+I*mu)/ je \] // where I is identity matrix.

mu, adaptive value is incremented by mu_inc (mu=mu+mu_inc) until reduced performance value is obtained. Then the network is changed and mu is decreased by mu_dec (mu=mu-mu_dec). To calculate the Jacobian jX, the parameter mem_reduc explain the use of speed and memory. Higher values of mem_reduc decrease the memory and increase training times. If mem_reduc is 1, then trainlm require a lot of memory and will execute faster. If mem_reduc to 2 memory requirement drop by a factor of two, but training time is reduced. If any one of these conditions are true then the training stops. The conditions are: maximum time exceeded, maximum epochs reached, performance gradient < min_grad or the mu > mu_max, performance is minimized to the goal or the validation performance > max_fail times.

In Neural Network tool box **learngdm** is gradient descent with bias learning function and momentum weight. Its syntax is

\[ [dW1,LS1] = learngdm(W1,P1,Z1,N1,A1,T1,E1,gW1,gA1,D1,LP1,LS1) \]

info = learngdm(code)

\[ [db,LS1] = learngdm(b1,ones(1,Q1),Z1,N1,A1,T1,E1,gW1,gA1,D1,LP1,LS1) \]

The input that learngdm uses are W1 denotes weight matrix of dimension S1 x R1 (or S1 x 1 bias vector), P1 denotes R1 x Q1 input vectors, Z1 denotes S1 x Q1 weighted
input vectors, \(N1\) denotes \(S1 \times Q1\) net input vectors, \(A1\) denotes \(S1 \times Q1\) output vectors, \(T1\) denotes \(S1 \times Q1\) layer target vectors, \(E1\) denotes \(S1 \times Q1\) layer error vectors, \(gW1\) denotes gradient \(S1 \times R1\), \(gA1\) denotes output gradient \(S1 \times Q1\), \(D1\) denotes \(S1 \times S1\) neuron distances, \(LP1\) denotes Learning parameters, \(LS1\) denotes learning state, initially it should be = [] and returns \(dW1\) which is \(S1 \times R1\) weight change matrix and \(LS1\) is new learning state. Learning occurs according to following default values of learnngdm's learning parameters:

<table>
<thead>
<tr>
<th>Learning Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP.lr</td>
<td>0.01</td>
<td>Rate of Learning</td>
</tr>
<tr>
<td>LP.mc</td>
<td>0.9</td>
<td>Momentum constant</td>
</tr>
</tbody>
</table>

Table 5.2: Default values of learnngdm's learning parameters

The information that learnngdm (code) returns, where pdefaults is default learning, parameter pnames is learning parameters, needg returns 1 if this function uses \(gW1\) or \(gA1\). learnngdm needs these values to compute a weight change. The default learning state given by initial values is: \(ls = []\); \([dW,ls] = learnngdm([],[],[],[],[],[],gW,[][],lp,ls)\)

New learning state and the weight change are returned by learnngdm. Gradient descent with momentum is given by: \(dW = (1-mc)*lr*gW + mc*dWprev\). learnngdm calculates \(dW, E, W, LR\) and \(MC\), where \(dW\) denotes the weight change from the neuron's input \(P\) for a given neuron, \(E\) denotes the error, \(W\) denotes the weight (or bias), \(LR\) denotes learning rate and \(MC\) denotes momentum constant. From the learning state \(LS\) the \(dWprev\) previous weight change is stored and read.
e) Neural Network is shown in figure given

![Figure 5.7: Neural Network](image1)

f) Training the neural network

![Figure 5.8: Training of Neural Network](image2)
g) Training Result

Figure 5.9: Training result of Neural Network

h) Simulating the neural network

Figure 5.10: Simulation of Neural network
4. Analysis of result obtained.

MATLAB is used for detecting change in the structure. The algorithm is implemented in Neural Network tool box. The Neural Network tool box is used for creating neural network. Assign input vector (v) values \( v = \{0 \ 0 \ 1 \ 1 \ 0 \; 0 \ 1 \ 0 \ 0 \ 0 \} \) and assign target values \( t = \{0 \ 1 \ 1 \ 10\} \). The neural network is created with name Changedetection along with the following Network properties: the network type is Feed forward Back propagation with Input Range \([0,1;01]\) and TRAINLM as the Training Function. The Adaptation Learning Function is LEARNGDM and Performance function as MSE with No of Layers is 2. The neural network is trained and the result of training is obtained. The neural network is then simulated. The result is analyzed as ‘1’ in the output vector will indicate the change in content and ‘0’ in the output vector will corresponds to no change in content.
5.6.2 Method Followed For Detecting Change In The Content

1. Initially the web graph is derived from web pages.

2. Obtain content vector V1 for graph G1 at any time t and content vector V2 for the same graph G1 at any time t + t1:
   \[ V1 = (0, 0, 1, 1, 0) \]
   \[ V2 = (0, 1, 0, 0, 0) \]

3. MATLAB is used for detecting change in the Content
   a) The Neural Network tool box is used for creating neural network
   b) Assign input vector (v) values

Figure 5.12: Assign Input values to neural Network
c) Assign target values (t)

Figure 5.13: Assign target values to Neural Network

d) Create neural network with following:-

Figure 5.14: Neural Network description
i) Name: Changedetection

ii) Network properties

1) Network Type: Feed forward Back propagation

2) Input Range: [0,1;01]

3) Training Function: TRAINLM

4) Adaptation Learning Function: LEARNGDM

5) Performance function: MSE

6) No of Layers: 2

e) Neural Network is shown in figure given below

Figure 5.15: Neural Network
f) Training the neural network

Figure 5.16: Training of Neural Network

g) Training Result

Figure 5.17: Training result of Neural Network
h) Simulating the neural network

![Simulation of Neural Network](image)

Figure 5.18: Simulation of Neural Network

4. Analysis of result obtained.

![Results of Neural Network](image)

Figure 5.19: Results of Neural Network

\[ R = (0, 1, 1, 0, 0) \]

Obtaining content vector \( V_1 \) for graph \( G_1 \) at any time \( t \) and content vector \( V_2 \) for the same graph \( G_1 \) at any time \( t + t_1 \): \( V_1 = (0, 0, 1, 1, 0) \), \( V_2 = (0, 1, 0, 0, 0) \)
MATLAB is used for detecting change in the Content. The Neural Network tool box is used for creating neural network. Assign input vector \( (v) \) values \( v=\{0,0,1,1,0;0,1,0,0,0\} \) and Assign target values \( (t) \) \( t=\{0,1,1,1,0\} \). The neural network is created with the name Changedetection and with the following Network properties: the Network Type is Feed forward Back propagation with Input Range \([0,1;01]\) and TRAINLM as the training Function. The Adaptation Learning Function is LEARNGDM and MSE as Performance function with number of layer as 2. The neural network is trained and the result of training is obtained. The neural network is then simulated. The result is analyzed as ‘1’ in the output vector will indicate the change in structure and ‘0’ in the output vector will corresponds to no change in structure. In migrating parallel web crawler, neural network based change detection method detect for changes in content and structure yield high quality fresh pages are downloaded.

<table>
<thead>
<tr>
<th>Crawling Technique</th>
<th>Running Time in Seconds</th>
<th>Active threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional Crawler</td>
<td>56</td>
<td>4</td>
</tr>
<tr>
<td>Single threaded Crawler</td>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>Agent Based Crawler</td>
<td>62</td>
<td>3</td>
</tr>
<tr>
<td>Migrating Parallel Crawler</td>
<td>24</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5.3: Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Running time and Active Threads
Figure 5.20(a): Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of running time

Figure 5.20(b): Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Active Threads

110
<table>
<thead>
<tr>
<th>Crawling Technique</th>
<th>Number of Link Tested</th>
<th>Number of Link in Queue</th>
<th>Total Number of Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional Crawler</td>
<td>12</td>
<td>46</td>
<td>58</td>
</tr>
<tr>
<td>Single threaded Crawler</td>
<td>10</td>
<td>48</td>
<td>58</td>
</tr>
<tr>
<td>Agent Based Crawler</td>
<td>16</td>
<td>42</td>
<td>58</td>
</tr>
<tr>
<td>Migrating Parallel Crawler</td>
<td>24</td>
<td>34</td>
<td>58</td>
</tr>
</tbody>
</table>

Table 5.4: Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Link Tested and Link in Queue
<table>
<thead>
<tr>
<th>Crawling Technique</th>
<th>Number of Pages Visited</th>
<th>Pages Visited / Sec</th>
<th>Number of changes detected</th>
<th>Percentage of Change Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional Crawler</td>
<td>21456</td>
<td>14</td>
<td>5001</td>
<td>23.30817</td>
</tr>
<tr>
<td>Single threaded Crawler</td>
<td>19456</td>
<td>7</td>
<td>5052</td>
<td>25.96628</td>
</tr>
<tr>
<td>Agent Based Crawler</td>
<td>31453</td>
<td>28</td>
<td>10060</td>
<td>31.98423</td>
</tr>
<tr>
<td>Migrating Parallel Crawler</td>
<td>42452</td>
<td>35</td>
<td>18535</td>
<td>43.66108</td>
</tr>
</tbody>
</table>

Table 5.5: Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Pages Visited, Pages Visited/Sec and Changes Detected

Figure 5.22(a): Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Pages Visited
Figure 5.22(b): Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Pages visited/Sec

Figure 5.22(c): Comparison of Migrating Parallel Web Crawlers with other Crawlers on the basis of Changes Detected
The table 5.3 provided the experimental statistics of running time and number of active threads; and the figure 5.20(a) and figure 5.20(b) demonstrated the graph that gives comparison of Migrating Parallel Crawlers with other Crawlers on the basis of running time in seconds and number of Active Threads respectively. The table 5.4 provided experimental statistics of link tested and link in queues; and the figure 5.21 demonstrated the graph that gives the comparison of Migrating Parallel Crawlers with other Crawlers on the basis of Link Tested and Link in Queue. The table 5.5 provided experimental statistics of pages visited, pages visited/sec and changes detected and the figure 5.22(a), figure 5.22(b), figure 5.22(c) demonstrated the graph that shows the Comparison of proposed Migrating Parallel Web Crawlers with other crawlers on the basis of pages visited, pages/sec and changes detected respectively. The table 5.6 provided the experimental statistics and the figure 5.23 demonstrated the graph that gives the comparison of Migrating Parallel Crawlers with other crawlers on the basis of percentage changes detected. The code is generated with help of MATLAB Compiler. The above module is integrated with the crawler.
5.7 Conclusion

In this chapter a change detection method based on neural network in migrating parallel web crawler is proposed and implemented that detect for changes in content and structure and yield high quality fresh pages. The Neural Network tool box is used for the implementation. The proposed approach is easy to implement, reliable, cost effective, less time to perform and fault tolerant in nature. The results obtained using this approach are always appreciable as compared to other techniques and provide support to effective Migrating Parallel Web Crawler approach that detects changes in the content and structure using neural network.